Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Recomendação de Técnicas de Pré-Processamento por Meta-Aprendizado no Contexto de AutoML

Juliana Mayumi Hosoume

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Luís Paulo Faina Garcia

Brasília
2020

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Recomendação de Técnicas de Pré-Processamento por Meta-Aprendizado no Contexto de AutoML

Juliana Mayumi Hosoume

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Luís Paulo Faina Garcia (Orientador)
Universidade de Brasília

Prof. Dr. Thiago Paulo Faleiros
Universidade de Brasília

Prof. Dr. Vinícius Ruela Pereira Borges
Universidade de Brasília

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 9 de dezembro de 2020

Dedicatória

Para Jonathan, Eidy e Célia. Muito obrigada!

Agradecimentos

Sou grata a Deus acima de tudo.

Agradeço especialmente ao Jonathan por todo o companheirismo. Pelo carinho e atenção. Por sempre estar presente e me escutar. Obrigada por tudo!

Grata pelos pais maravilhosos. Pela confiança e suporte inestimáveis. Muito obrigada pelo suporte incondicional.

Agradeço à Universidade de Brasília e seu incrível corpo docente. Obrigada por todo conhecimento compartilhado e por todo esforço empreendido para minha formação.

Ao Dr. Luís Paulo Faina, meu orientador, agradeço por sua dedicação e paciência imensuráveis. Obrigada por sempre estar disposto a ouvir. Sou muito grata por todo tempo dedicado para escutar minhas indagações. Sua ajuda e orientação foram indispensáveis para esse trabalho.

Grata aos membros da banca por cederem tempo para leitura do trabalho e pela disposição em contribuir com ele.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

O aprendizado de máquina possui uma multitude de algoritmos e técnicas de pré-processamento que tratam de problemas de classificação. Combiná-los para projetar a melhor sequência de técnicas, ou *pipeline*, de aprendizado de máquina é uma tarefa desafiadora. Diferentes abordagens já foram investigadas, entre elas processos manuais de construção de *pipelines*, até utilização da otimização Bayesiana e de programação genética. No entanto, cada uma destas abordagens tem impedimentos, como a necessidade de um especialista humano ou elevado custo computacional. O meta-aprendizado pode superar estes dificuldades por meio do conhecimento acumulado em experimentos anteriores. Assim, o conhecimento é armazenado em uma meta-base cujos atributos preditivos são meta-características extraídas de conjuntos de dados, e os atributos alvo representam o desempenho preditivo de *pipelines* bem sucedidos aplicados a esses conjuntos de dados. Este estudo propõe o uso do meta-aprendizado como base para desenho de um processo iterativo de construção de *pipelines* para problemas de classificação. Nesse sentido, o sistema proposto é responsável por predizer um conjunto diversificado de algoritmos de desbalanceamento e filtros de ruído. Para tal, foi utilizada uma meta-base composta de 130 meta-características e quase 400 conjuntos de dados para induzir meta-regressores com diferentes vieses. O sistema de recomendação possui duas fases, uma *on-line* e uma *off-line*. Na fase *on-line* do sistema de recomendação de *pipelines*, as meta-características são extraídas de um novo conjunto de dados. Elas são então utilizadas como entrada de meta-regressores que predizem a acurácia balanceada de uma combinação de um pré-processador e um classificador. Como qualquer número de algoritmos de pré-processamento pode ser aplicado posteriormente, muitas etapas de predição são realizadas sequencialmente até que nenhuma técnica de pré-processamento seja recomendada. Em cada etapa, as meta-características são extraídas deste novo conjunto de dados pré-processado, e todos os desempenhos para cada combinação são preditos. Se uma técnica de pré-processamento for recomendada, este algoritmo é aplicado ao conjunto de dados, e o processo é repetido iterativamente. Na fase *off-line*, o meta-conjunto de meta-dados, uma coleção de meta-características com acurácia balanceada de cada combinação de algoritmos e classificadores de pré-processamento disponíveis, pode ser incrementado pela introdução do desempenho calculado e das meta-características do novo conjunto de dados. Cada meta-regressor é então atualizado ou retreinado. No contexto das combinações selecionadas no papel, 40 meta-modelos são gerados para prever cada desempenho. Na avaliação do sistema, foram analisadas as quatro etapas do meta-aprendizado: a meta-base, o nível meta, o nível base e o nível de construção dos *pipelines*. Os resultados foram comparados a dois *baselines*, o aleatório, no qual o *pipeline* é construído de ao acaso, e o padrão, no qual o *pipeline* com melhor desempenho na média é sempre selecionado. Os meta-regressores previram a precisão balanceada das combinações com baixo erro, e alguns superaram os *baselines*. De acordo com os resultados experimentais, a estratégia proposta teve melhor desempenho do que as *baselines*.

Palavras-chave: Aprendizado de Máquina, Meta-aprendizado, AutoML, Recomendação, *Pipeline*

Abstract

Machine learning has a multitude of algorithms and preprocessing techniques that address classification problems. Combining them to design the best data classification pipeline is a challenging task. Different approaches have already been investigated, including handmade pipelines, Bayesian optimization and genetic programming. Nevertheless, each of these approaches has hindrances, such as the need of a human specialist for handmade pipelines, or the computational cost of Bayesian optimization, and genetic programming. Meta-learning can overcome these drawbacks through knowledge about pipelines accumulated from previous experiments. Thus, the knowledge is stored in a meta-base whose predictive attributes are meta-features extracted from datasets, and the target attributes represent the predictive performance of successful pipelines applied to these datasets. This study proposes the use of meta-learning as a pipeline builder to predict the performance of combinations of preprocessing techniques, like noise detection and unbalanced algorithms for classification problems. For such, a meta-base composed of 130 meta-features and almost 400 datasets were used to induce meta-regressors with different biases. The recommendation system has two phases, an on-line and an off-line. In the on-line phase of the recommendation system of pipelines, the metafeatures are extracted from a new data set. They are then used as input meta-regressors that predict the balanced accuracy of a combination of a pre-processor and a classifier. As any number of preprocessing algorithms can be applied later, many prediction steps are performed sequentially until no preprocessing technique is recommended. At each step, meta-features are extracted from this new preprocessed data set, and all performances for each combination are predicted. If a pre-processing technique is recommended, this algorithm is applied to the data set, and the process is repeated iteratively. In the off-line phase, the meta-data set, a collection of meta-characteristics with balanced accuracy of each combination of available preprocessing algorithms and classifiers, can be enhanced by introducing the calculated performance and meta-characteristics of the new data set. Each meta-regressor is then updated or re-trained. In the context of the selected combinations on paper, 40 meta-models are generated to predict each performance. In the system evaluation, the four steps of the meta-learning were analyzed: the meta-base, the meta level, the base level and the construction level of the pipelines. The results were compared to two baselines, the random baseline, in which the pipeline is constructed according to chance, and the default baseline, in which the pipeline with the best performance on average is always selected. The meta-regressors predicted the balanced accuracy of the combinations with low error, and some outperformed the baselines. According to the experimental results, the proposed strategy performed better than the baselines.

Keywords: Machine Learning, Meta-Learning, AutoML, Recommendation, Pipeline

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AM Aprendizado de Máquina.

ANN *Artificial Neural Network*.

AutoML Aprendizado de Máquina Automatizado ou *Automated Machine Learning*.

CART *Classification and Regression Tree (Decision Tree)*.

DF *Default Baseline* ou *Baseline* de Melhor Desempenho Médio.

DT Árvore de Decisão ou *Decision Tree*.

DWNN *Distance Weighted k-Nearest Neighbour*.

ENN *Edited Nearest Neighbours*.

GB *Gradient Boosting*.

GNB *Gaussian Naive Bayes*.

HARF *High Agreement Random Forest*.

IA Inteligência Artificial.

k-NN *k-Nearest Neighbours*.

LD *Linear Discriminant*.

LR *Logistic Regression*.

MD Mineração de Dados.

MSE *Mean Square Error*.

MtL Meta-Aprendizado ou *Meta-Learning*.

RD *Random Baseline* ou *Baseline* Aleatório.

RF *Random Forest*.

RFR *Random Forest Regressor*.

RMSE *Root Mean Square Error*.

RU *Random Undersampling*.

SMOTE *Synthetic Minority Over-sampling*.

SVM *Support Vector Machine*.

SVR *Support Vector Regressor*.

Capítulo 1

Introdução

O primeiro trabalho reconhecido da área de Inteligência Artificial (IA) foi realizado por McColloch e Pitts em 1943 [2]. Apenas em 1957 Solomonoff publicou o primeiro artigo sobre Aprendizado de Máquina (AM) [3]. Desde então, há crescimento significativo em pesquisas e aplicações de técnicas dessa área. Enquanto as publicações em Ciência da Computação cresceram em torno de cinco vezes desde 1996, houve aumento de mais de sete vezes de publicações na área de IA nesse período [4].

Há uma miríade de aplicações diretas ou indiretas de técnicas de AM. Os problemas resolvidos pela AM são das mais diversas áreas como na saúde, para suporte em diagnósticos médicos, processamento de sinais biomédicos, interpretação de microscopias e radiografias [5], bem como na predição de doenças a exemplo de Alzheimer [6], alguns tipos de câncer [7] e transtorno de estresse pós-traumático [8]. Ademais, AM também é empregado em soluções de problemas na construção civil para monitoramento e detecção de danos de grandes prédios [9], controle de trânsito [10] e até mesmo na própria computação, para detecção de intrusão [11] e teste de *software* [12], por exemplo.

Ao longo dos anos, o número de conjuntos de dados disponíveis aumentou, concomitantemente ao poder computacional. Neste cenário, as análises complexas de dados foram facilitadas. No entanto, definir os melhores métodos a serem aplicados em um conjunto de dados específico ainda é uma tarefa exaustiva, e geralmente envolve o uso de *pipelines* de AM. A definição de um *pipeline*, ou sequência de técnicas de AM para solução de uma tarefa, requer não apenas a escolha de um algoritmo de aprendizagem adequado, como também a seleção de técnicas de pré-processamento. Além disso, cada uma destas etapas é sensível aos dados, ou seja, o desempenho depende das características do conjunto de dados [13, 14].

Um *pipeline* em AM é geralmente composto de cinco etapas: coleta de dados, preparação e pré-processamento, seleção de um algoritmo de reconhecimento de padrões adequado e, finalmente, treinamento e avaliação de um modelo [15]. Encontrar um *pipe-*

line adequado para um novo problema de AM depende de experiência e é muito custoso computacionalmente e temporalmente. Um sistema de Aprendizado de Máquina Automatizado ou *Automated Machine Learning* (AutoML) poderia ajudar não especialistas nesta tarefa [16]. Diferentes abordagens permitem esta busca, como a programação genética [17] e a otimização Bayesiana [18], e meta-aprendizado [19]. No entanto, cada um deles possui desvantagens, como necessidade de especialista e elevado custo computacional. O Meta-Aprendizado ou *Meta-Learning* (MtL) pode superar estes entraves com o uso de meta-conhecimento [19].

Neste contexto, o MtL aborda o problema da seleção de algoritmos pelo uso do conhecimento acumulado de tarefas anteriores, ou meta-conhecimento [20]. Esta abordagem tem sido empregada em sistemas de recomendação [21], sistemas de Mineração de Dados (MD) com agentes inteligentes [22] e otimização de hiper-parâmetros [23]. Embora tenham sido obtidos bons resultados na predição do desempenho de um classificador baseado em características estatísticas e geométricas de um conjunto de dados [24], menos estudos focam no pré-processamento de dados [25, 26, 27].

O MtL vai além do aprendizado base tradicional [28]. Comumente, o primeiro passo na utilização de MtL consiste na construção de uma meta-base composta de meta-exemplos [29]. Cada meta-exemplo inclui meta-características e um rótulo para cada algoritmo de AM utilizado no estudo. As meta-características são um conjunto de características extraídas dos dados. Por outro lado, o rótulo pode ser uma medida do desempenho de um modelo de base.

Após a construção de um conjunto de meta-dados com meta-exemplos suficientes, o próximo passo é a indução de um meta-modelo. Este meta-modelo pode ser selecionado a partir de um conjunto de algoritmos ML candidatos. O objetivo do meta-modelo é prever o comportamento de um modelo de base para um novo conjunto de dados, usando apenas as meta-características extraídas. Portanto, além de apontar o algoritmo de melhor desempenho preditivo para o conjunto de dados, um ranqueamento ou lista de medidas de desempenho pode ser obtida pela aplicação do meta-modelo [20]. Entretanto, ocasionalmente o meta-modelo não é suficientemente preciso. Trabalhos anteriores sugerem a necessidade de uma etapa de pré-processamento, como o tratamento do desequilíbrio de dados [30].

Nesse contexto, a utilização de uma abordagem de MtL em AutoML para a recomendação de múltiplas técnicas de pré-processamento complexas é promissora. Portanto, a hipótese é de que *pipelines* compostos de múltiplos passos de técnicas de pré-processamento e um classificador escolhido de um conjunto com vieses diversos, pode ser construído por um sistema baseado em experiências anteriores. O uso de MtL pode ser propício para reduzir esforços e custos temporais, além de permitir uma melhora incremental do sistema.

Neste trabalho, um projeto de recomendação de *pipelines* para AutoML é investigado. Para as técnicas de pré-processamento, o foco está nos algoritmos de detecção de ruído e técnicas para problemas desbalanceados. Neste sentido, uma abordagem de MtL é utilizada para resolver a complexa tarefa de encontrar uma combinação adequada de múltiplas técnicas de pré-processamento e um classificador para um novo problema AM. Para tal, o estado-da-arte em meta-características [31] são usadas para induzir um meta-regressor adequado para prever o desempenho dos *pipelines*. Uma análise de acerto repetida em várias etapas de recomendação é aplicada para avaliar a construção do *pipeline*. Finalmente, os resultados são comparados com *baselines*, a fim de avaliar a eficiência da metodologia proposta. Nos resultados experimentais, uma recomendação baseada no regressor *Random Forest* (RF) foi melhor do que a *baseline* padrão, tanto na análise base como na construção dos *pipelines*.

Este trabalho está organizado em cinco capítulos. Este, Capítulo 1, introduz os conceitos a serem abordados, assim como a hipótese central. O Capítulo 2 apresenta e discute os estudos relacionados e os antecedentes teóricos em pré-processamento, meta-aprendizagem e aprendizagem automática da máquina. A metodologia e procedimentos utilizados para este trabalho experimental são evidenciados no Capítulo 3. Além disso, nesse mesmo capítulo são apresentados o desenho do sistema nas fases *off-line* e *on-line*. O Capítulo 4 detalha os resultados dos experimentos. O Capítulo 4 também discute o resultado dos experimentos. Finalmente, o Capítulo 5 conclui o trabalho, apresentando perspectivas e limitações do sistema proposto.

Capítulo 2

Revisão da Literatura

2.1 Aprendizado de Máquina

A Inteligência Artificial (IA) pode ser definida de múltiplas formas, de acordo com o autor e sua especialidade. A IA pode ser entendida como a incorporação da inteligência humana em máquinas ou como a medida com que um agente artificial consegue atingir suas metas em diferentes contextos [32]. Nesse sentido, o campo de Aprendizado de Máquina (AM) está inserido na IA, sendo, portanto, a forma fundamental para um computador, ou um agente artificial, tornar-se inteligente [33].

Uma definição comum de AM está centrada no estudo sistemático de algoritmos e sistemas os quais, por meio de experiências, conseguem melhorar seu desempenho em uma determinada tarefa. As experiências podem ser um conjunto de informações ou dados fornecidos para o agente computacional. Em geral, essas experiências possuem um impacto significativo para o sucesso do processo de aprendizagem [34].

A partir do conjunto de dados fornecidos, as informações podem ser coletadas e incorporadas à base de informações e, assim, gerando um modelo. Esse modelo é então utilizado para predição de resultados com base em novos dados ou até mesmo para tomadas de decisão. O desempenho do sistema de aprendizado depende fortemente da quantidade e qualidade das informações fornecidas, assim como do algoritmo escolhido [34].

Nota-se que o objetivo principal é encontrar a função descritora do fenômeno em estudo, no entanto, apenas com os exemplos de treino disponíveis. Quando não há informações adicionais sobre o sistema, assume-se a hipótese de aprendizado indutivo. O algoritmo de aprendizado inicia com a definição de um modelo a partir do conjunto de treino. Nesse contexto, qualquer modelo ou hipótese que se aproxima da função descritora e é construído a partir de um conjunto suficientemente grande de exemplos de treino, também se aproximará da descrição de exemplos não observados [34].

Há uma busca pelo melhor algoritmo de aprendizado, tanto na rapidez de aprendizado quanto na adaptação aos mais diferentes contextos e características dos conjuntos de dados de entrada. No entanto, a eficiência de um algoritmo de aprendizado em particular depende do domínio das informações fornecidas, portanto a generalização não é facilmente obtida [35]. Esse problema de escolha é conhecido como seleção de modelo ou seleção de algoritmo [29].

Os algoritmos de aprendizado diferenciam-se pela maneira com a qual encontram os padrões nos dados fornecidos. Há, por conseguinte, um viés associado a cada algoritmo, de modo que, com o mesmo conjunto de exemplos de treino, diferentes algoritmos geram resultados distintos. Assim, para que uma predição seja feita com o conjunto de restrito de dados de treino, suposições são realizadas para escolha da forma de generalização dos dados. Esse conjunto de suposições constitui o viés indutivo ou viés de aprendizado [21, 34]. O viés indutivo pode ser dividido em dois tipos de vieses diferentes, o viés representacional e o viés procedural.

O viés representacional são os estados possíveis de serem alcançados pelo algoritmo no espaço de busca, comumente o espaço de hipóteses. O espaço de todos os possíveis modelos é chamado de espaço de hipóteses H. Desse modo, a tarefa de aprendizagem pode ser vista como a busca de uma hipótese $h \in H$, tal que h se aproxima da função alvo c em um espaço de instâncias ou espaço de observações X. O viés representacional também é chamado de viés de linguagem [36, 21].

O viés procedural, ou viés algorítmico, está associado com a ordem da mudança dos estados no espaço definido pelo viés representacional. À medida que há refinamento pelo treino, uma nova hipótese deve ser escolhida no espaço de hipóteses, de maneira que são feitas suposições pelo algoritmo para melhor decisão por meio de heurísticas.

2.1.1 *No Free Lunch*

O teorema *No Free Lunch* (em tradução livre, "não há almoço grátis") foi formalizado no contexto de busca [13], AM [35] e otimização [37]. O grande impacto desse teorema é concretizado na Lei da Conservação para Generalização de Desempenho, a qual afirma que qualquer desempenho positivo de um sistema de aprendizagem em um contexto deve ser compensando pela perda de desempenho em contextos distintos [38]. Em conclusão, não há a possibilidade de criar uma estratégia de aprendizado universal, com bom desempenho de forma geral.

Ainda que o teorema *No Free Lunch* seja um argumento contra a pesquisa por algoritmo de aprendizado para resolução de problemas gerais, na condição de distribuição uniforme das funções modeladoras dos fenômenos, as afirmações do teorema não discorrem sobre medidas de desempenho esperadas para algoritmos sob um novo contexto. Por

conseguinte, pode ser investigado quão bem um determinado algoritmo irá desempenhar ao serem apresentadas novas experiências [39]. Desse modo, para encontrar o algoritmo mais adequado para um determinado problema podem ser usadas a seleção de modelos por avaliação por utilização de *Cross-Validation*, projeto de um modelo por especialista ou meta-aprendizado.

2.2 Aprendizado de Máquina Automatizado

O Aprendizado de Máquina Automatizado ou *Automated Machine Learning* (AutoML) está relacionado com os processos de retirada da responsabilidade de escolha de algoritmos e hiper-parâmetros do usuário, de modo que a máquina encontre a melhor seleção de modelos e otimização de hiper-parâmetros. Em [16] são listadas tarefas de Aprendizado de Máquina (AM) que podem ser automatizadas. Entre elas, nas fases iniciais de manipulação de dados estão leitura e formatação de dados, detecção e manipulação de dados enviesados e com valores faltantes e extração de características relevantes. A automatização também abrange a fase de aprendizado como a escolha do algoritmo mais adequado para o problema, obtenção de novos dados por aprendizado ativo, criação de conjuntos de treino, validação e teste apropriados e seleção de algoritmos que satisfaçam as restrições de recursos para treino e execução. Por fim, também podem ser considerados exemplos possíveis de automatização as tarefas de execução de meta-aprendizado, transferência de conhecimento e geração de relatórios explicativos.

Definir uma sequência ideal de métodos para a Mineração de Dados (MD) não é uma tarefa simples. Há muitos algoritmos diferentes de pré-processamento e AM que podem ser combinados para implementar uma solução de AM com um bom desempenho preditivo para um novo conjunto de dados. No entanto, como declarado nos teoremas *No Free Lunch* [37, 14] (Seção 2.1.1), nenhum algoritmo possui o melhor desempenho em todos os problemas de AM, por conseguinte muito tempo e esforço são exigidos. O processo de AutoML visa facilitar a tarefa de selecionar *pipelines* adequados e melhorar a produtividade.

Um sistema AutoML pode se concentrar em diferentes problemas de AM. Alguns estudos visam à seleção e geração de recursos [40, 41] como um passo chave para o sucesso de uma solução AM. Outros exploram o problema de seleção de modelos e a etapa de otimização do hiper-parâmetros [42, 43]. A recomendação de um fluxo de trabalho completo também é investigada em estudos recentes [17, 19]. A definição de um *pipeline* completo é uma tarefa complexa. O espaço de busca de estruturas e de algoritmos candidatos para compor são extensos e a escolha dos componentes requer experiência específica.

Além disso, o desempenho dos algoritmos é sensível à entrada, de modo a aumentar a complexidade da tarefa [19].

Muitas abordagens diferentes para implementação de AutoML foram propostas. Entre elas está a programação genética, utilizada no TPOT [17], um método baseado na geração e evolução de *pipelines*. Alternativamente, a Auto-sklearn [23] e a AutoWEKA [18] empregam a otimização Bayesiana, que é centrada em modelos probabilísticos para prever o desempenho de algoritmos. Uma outra opção para resolver o problema da busca de um *pipeline* é utilizar o MtL, como aplicado no RankML [19].

Neste trabalho, o método proposto estende estudos anteriores sobre a abordagem MtL em AutoML [44], considerando a recomendação de múltiplos passos de pré-processamento do *pipeline*. Além disso, ele difere do sistema RankML [19] e AutoML [23] no processo de avaliação e construção de *pipelines* adequados. Por um lado, o RankML considera cada *pipeline* como um grafo acíclico e utiliza a representação da topologia *pipelines* como sequências de palavras. *Hashes* são criados para identificar cada *pipeline*. Já o AutoML aplica o MtL combinado com a otimização Bayesiana. Neste estudo, os *pipelines* são construídos de forma iterativa, baseado apenas na meta-caracterização dos conjuntos de dados, sem armazenar informações sobre a estrutura do *pipeline* na meta-base. Portanto, um possível custo computacional de processamento da estrutura do *pipeline* é reduzido, visto que a representação do problema investigado é simplificada. Além disso, o foco é dado para os passos de pré-processamento necessários.

2.3 Meta-aprendizado

O prefixo *meta* tem origem na linguagem grega e é atribuído a um conceito ou tema que referencia ou analisa a si mesmo [1]. O meta-aprendizado, em especial no campo do AM, é aprender a tarefa de aprendizado. Em outras palavras, como definido por [20], é a busca por entender como utilizar o meta-conhecimento para construir ou melhorar sistemas de aprendizado por adaptação de processos de AM e de mineração de dados. Já meta-conhecimento pode ser entendido como informações e dados obtidos de processos de aprendizado anteriores. Comumente essas informações consistem em propriedades mensuráveis de *datasets* e algoritmos de aprendizado, combinadas com propriedades aferidas dos modelos criados a partir dos algoritmos de aprendizado [45]. Inicialmente idealizado por Aha et al. [46], a abordagem MtL está centrada em aprender o resultado através do conhecimento de tarefas já conhecidas. Para tal, MtL é baseado nos resultados de algoritmos e técnicas, não no viés ou diferenças internas dos algoritmos de aprendizagem [22].

[1] `https://dicionario.priberam.org/meta`

O meta-aprendizado, portanto, auxilia no problema de seleção de algoritmos e modelos por meio da utilização de experiências passadas. Diferentemente do escopo de aprendizado tradicional, no meta-aprendizado o acúmulo de experiências não é focado em uma tarefa, todavia está focado no desempenho de diversas aplicações de sistemas de aprendizagem. Essa área possui início formal na década de 70, com a publicação de Rice [47] e Maudsley [48] e continua a atrair interesse da comunidade de AM [49].

O MtL se encaixa no modelo de seleção de algoritmos do Rice [47]. Nesta abstração, há quatro componentes principais: o espaço do problema P, o espaço de características F, o espaço de algoritmo A e o espaço de desempenho Y. Eles representam o conjunto de possíveis instâncias do problema, uma coleção de caracterizações mensuráveis das instâncias, um conjunto escolhido de algoritmos adequados que poderiam resolver o problema e métricas de desempenho para cada combinação de algoritmos e instâncias, respectivamente. Um sistema que emprega MtL é baseado em um procedimento para mapear um problema $p \in P$, definido por meta-características $f \in F$, para um algoritmo $\alpha \in A$, que é o algoritmo mais adequado para a tarefa de acordo com a métrica $y \in Y$, $y(\alpha(p))$ [50].

Formalmente, o meta-aprendizado pode ser caracterizado como o mapeamento de algoritmos com tarefas de classificação. Portanto, é induzido um meta-modelo a partir de um conjunto de treino $\{< f(p), y_A(p) >: p \in P\}$, tal que, para uma tarefa p de um conjunto P de tarefas de classificação, $p \in P$, dada uma caracterização de p, $f(p)$, por um mecanismo fixo, tem-se que $y_A(p)$ é um algoritmo pertencente a A com melhor desempenho para p [39].

Neste contexto, a complexidade destes problemas pode ser medida pela extensão e dimensionalidade de cada conjunto e pela complexidade dos mapeamentos. Por exemplo, o espaço de problemas P pode conter tarefas diferentes de vários domínios. Idealmente, este conjunto deveria ser composto de tantos problemas diversos quanto o sistema MtL seria apresentado. Estes conjuntos de dados de problemas podem ser coletados da literatura ou repositórios de dados abertos como UCI [51] ou OpenML [52].

2.3.1 Meta-características

O conjunto selecionado de meta-características (F) é fundamental para o sucesso do meta-modelo. Este conjunto deve conter informações suficientes sobre as tarefas investigadas. Além disso, um atributo indispensável de uma solução ML é a reprodutibilidade. Para este fim, a seleção de atributos preditivos significativos que permitem replicar todas as etapas de cálculo é fundamental para um sistema de recomendação baseado em MtL [1, 31].

Seguindo a definição formal apresentada por Alcobaça et al. [1] e Rivolli et al. [31], as meta-características são um conjunto de k valores extraídos de um conjunto de dados D. Portanto, este processo pode ser modelado como uma função $f : D \rightarrow \mathbb{R}^k$,

$f(D) = \theta(m(D, h_m), h_\theta)$. Esta equação destaca dois componentes principais das meta-características, as medidas de caracterização, m e as funções de sumarização, θ. Ambas dependem dos hiper-parâmetros h_m e h_θ, respectivamente. O método de cálculo de alguns grupos de medidas é definido por estes hiper-parâmetros, outros, como as meta-características simples, são calculados diretamente.

As meta-características podem ser divididas em cinco grupos: simples, teórico da informação, baseado em modelos, estatístico e *landmarking*[53, 31].

- As meta-características **simples**, também conhecidas como meta-características comuns, são as mais comumente utilizadas. Elas são facilmente e diretamente calculadas a partir dos conjuntos de dados, portanto, requerem poucos recursos computacionais [53]. Algumas delas são o número de atributos, o número de atributos binários e o número de atributos categóricos.

- As meta-características de **teoria da informação** são medidas baseadas no cálculo da entropia e da teoria da informação [53].

- As meta-características **baseadas em modelo** são obtidas de um modelo de AM treinado nos dados. Normalmente, estas medidas são extraídas das propriedades dos modelos de Árvore de Decisão ou *Decision Tree* (DT) [53].

- Meta-características **estatísticas** avaliam os indicadores estatísticos dos dados. Consideramos aqui apenas as informações numéricas. Média, correlação, desvio padrão são alguns exemplos destas medidas estatísticas [53, 54].

- Meta-características de ***landmarking*** são baseadas no desempenho preditivo de um conjunto de algoritmos AM para caracterizar os dados. Para definir apropriadamente um conjunto de dados, estes algoritmos devem ter diversos vieses e baixo custo computacional [50].

Em algumas categorizações, há um sexto grupo composto de medidas não tradicionais de meta-características, como medidas de clusterização. Usualmente, estas meta-características são complexas, computacionalmente caras ou específicas do domínio [31]. No entanto, para algumas situações, elas têm mostrado resultados promissores [24, 55, 56]. Para este trabalho, as meta-características escolhidas estão listadas na Tabela 2.1.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Teoria da Informação	attr_conc_mean	Computa a média do coeficiente de concentração de cada par de atributos distintos.
Teoria da Informação	attr_conc_sd	Computa o desvio padrão do coeficiente de concentração de cada par de atributos distintos.
Teoria da Informação	attr_ent_mean	Calcula a média da entropia de Shannon para cada atributo preditivo.
Teoria da Informação	attr_ent_sd	Calcula o desvio padrão da entropia de Shannon para cada atributo preditivo.
Simples	attr_to_inst	Calcula a razão entre o número de atributos.
Landmarking	best_node_mean	Desempenho médio de um único nó de árvore de decisão.
Landmarking	best_node_mean_relative	Desempenho médio relativo de um único nó de árvore de decisão.
Landmarking	best_node_sd	Desvio padrão do desempenho de um único nó de árvore de decisão.
Landmarking	best_node_sd_relative	Desvio padrão relativo do desempenho de um único nó de árvore de decisão.
Estatística	can_cor_mean	Calcula a média das correlações canônicas dos dados.
Estatística	can_cor_sd	Calcula o desvio padrão das correlações canônicas dos dados.
Simples	cat_to_num	Calcula a razão entre o número de características categóricas e numéricas.
Clusterização	ch	Calcula o índice Calinski e Harabasz.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Teoria da Informação	class_conc_mean	Calcula a média do coeficiente de concentração entre cada atributo e classe.
Teoria da Informação	class_conc_sd	Calcula o desvio padrão do coeficiente de concentração entre cada atributo e classe.
Teoria da Informação	class_ent	Calcula o atributo alvo da entropia de Shannon.
Estatística	cor_mean	Calcula a média do valor absoluto da correlação dos pares de colunas do conjunto de dados distintos.
Estatística	cor_sd	Calcula o desvio padrão do valor absoluto da correlação dos pares de colunas do conjunto de dados distintos.
Estatística	cov_mean	Calcula a média do valor absoluto da covariância de pares de atributos de conjuntos de dados distintos.
Estatística	cov_sd	Calcula o desvio padrão do valor absoluto da covariância de pares de atributos de conjuntos de dados distintos.
Estatística	eigenvalues_mean	Computa a média dos valores próprios da matriz de covariância do conjunto de dados.
Estatística	eigenvalues_sd	Computa o desvio padrão dos valores próprios da matriz de covariância do conjunto de dados.
Landmarking	elite_nn_mean	Média do desempenho de um vizinho mais próximo da Elite.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Landmarking	elite_nn_mean_relative	Média relativa do desempenho de um vizinho mais próximo da Elite.
Landmarking	elite_nn_sd	Desvio padrão do desempenho de um vizinho mais próximo da Elite.
Landmarking	elite_nn_sd_relative	Desvio padrão relativo do desempenho de um vizinho mais próximo da Elite.
Teoria da Informação	eq_num_attr	Calcula o número de atributos equivalentes para uma tarefa preditiva.
Simples	freq_class_mean	Calcula a média da frequência relativa de cada classe distinta.
Simples	freq_class_sd	Calcula o desvio padrão da frequência relativa de cada classe distinta.
Estatística	g_mean_mean	Calcula a média geométrica de cada atributo.
Estatística	g_mean_sd	Calcula o desvio padrão da média geométrica de cada atributo.
Estatística	gravity	Calcula a distância entre o centro de massa das classes minoritária e majoritária.
Estatística	h_mean_mean	Computa a média harmônica de cada atributo.
Estatística	h_mean_sd	Computa o desvio padrão da média harmônica de cada atributo.
Simples	inst_to_attr	Calcula a razão entre o número de instâncias e atributos.
Clusterização	intt	Calcula o índice INT.
Estatística	iq_range_mean	Calcular a média do intervalo interquartílico (IQR) de cada atributo.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Estatística	iq_range_sd	Calcular o desvio padrão do intervalo interquartílico (IQR) de cada atributo.
Teoria da Informação	joint_ent_mean	Computa a média da entropia conjunta entre cada atributo e a classe.
Teoria da Informação	joint_ent_sd	Computa o desvio padrão da entropia conjunta entre cada atributo e a classe.
Estatística	kurtosis_mean	Calcula a média da curtose de cada atributo.
Estatística	kurtosis_sd	Calcula o desvio padrão da curtose de cada atributo.
Baseada em Modelo	leaves	Calcular o número de nós de folha no modelo DT.
Baseada em Modelo	leaves_branch_mean	Calcula a média do tamanho dos ramos no modelo DT.
Baseada em Modelo	leaves_branch_sd	Calcula o desvio padrão do tamanho dos ramos no modelo DT.
Baseada em Modelo	leaves_corrob_mean	Computa a média da corroboração das folhas do modelo DT.
Baseada em Modelo	leaves_corrob_sd	Computa o desvio padrão da corroboração das folhas do modelo DT.
Baseada em Modelo	leaves_homo_mean	Calcular a média da homogeneidade do modelo DT para cada nó folha.
Baseada em Modelo	leaves_homo_sd	Calcular o desvio padrão da homogeneidade do modelo DT para cada nó folha.
Baseada em Modelo	leaves_per_class_mean	Calcula a média da proporção de folhas por classe no modelo DT.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Baseada em Modelo	leaves_per_class_sd	Calcula o desvio padrão da proporção de folhas por classe no modelo DT.
Landmarking	linear_discr_mean	Médio do desempenho do classificador Linear Discriminante.
Landmarking	linear_discr_mean_relative	Médio relativa do desempenho do classificador Linear Discriminante.
Landmarking	linear_discr_sd	Desvio padrão do desempenho do classificador Linear Discriminante.
Landmarking	linear_discr_sd_relative	Desvio padrão relativa do desempenho do classificador Linear Discriminante.
Estatística	mad_mean	Calcular a média do Desvio Mediano Absoluto (MAD) ajustado por um fator.
Estatística	mad_sd	Calcular o desvio padrão do Desvio Mediano Absoluto (MAD) ajustado por um fator.
Estatística	max_mean	Computa a média do valor máximo de cada atributo.
Estatística	max_sd	Computa o desvio padrão do valor máximo de cada atributo.
Estatística	mean_mean	Calcula a média do valor médio de cada atributo.
Estatística	mean_sd	Calcula o desvio padrão do valor médio de cada atributo.
Estatística	median_mean	Calcula a média do valor da mediana de cada atributo.
Estatística	median_sd	Calcula o desvio padrão do valor da mediana de cada atributo.
Estatística	min_mean	Calcula a média do valor mínimo de cada atributo.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Estatística	min_sd	Calcula o desvio padrão do valor mínimo de cada atributo.
Estatística	mut_inf_mean	Computa a média das informações mútuas entre cada atributo e o alvo.
Estatística	mut_inf_sd	Computa o desvio padrão das informações mútuas entre cada atributo e o alvo.
Landmarking	naive_bayes_mean	Média do desempenho do classificador Naive Bayes.
Landmarking	naive_bayes_mean_relative	Média relativa do desempenho do classificador Naive Bayes.
Landmarking	naive_bayes_sd	Desvio padrão do desempenho do classificador Naive Bayes.
Landmarking	naive_bayes_sd_relative	Desvio padrão relativa do desempenho do classificador Naive Bayes.
Baseada em Modelo	nodes	Calcula o número de nós não-folha no modelo DT.
Baseada em Modelo	nodes_per_attr	Calcula a proporção de nós por número de atributos no modelo DT.
Baseada em Modelo	nodes_per_inst	Calcula a proporção de nós não-folha por número de instâncias no modelo DT.
Baseada em Modelo	nodes_per_level_mean	Calcula a média da razão do número de nós por nível de árvore no modelo DT.
Baseada em Modelo	nodes_per_level_sd	Calcula o desvio padrão da razão do número de nós por nível de árvore no modelo DT.
Baseada em Modelo	nodes_repeated_mean	Calcula a média do número de nós repetidos no modelo DT.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Baseada em Modelo	nodes_repeated_sd	Calcula o desvio padrão do número de nós repetidos no modelo DT.
Simples	nr_attr	Calcula o número total de atributos.
Simples	nr_bin	Computa o número de atributos binários.
Simples	nr_cat	Computa o número de atributos categóricos.
Simples	nr_class	Computa o número de classes distintas.
Estatística	nr_cor_attr	Calcula o número de diferentes pares de atributos altamente correlacionados.
Estatística	nr_disc	Calcula o número de correlação canônica entre cada atributo e classe.
Simples	nr_inst	Computa o número de instâncias (linhas) no conjunto de dados.
Estatística	nr_norm	Calcula o número de atributos normalmente distribuídos com base em um determinado método.
Simples	nr_num	Computa o número de características numéricas.
Estatística	nr_outliers	Calcula o número de atributos com pelo menos um valor anterior.
Clusterização	nre	Calcula a entropia relativa normalizada.
Teoria da Informação	ns_ratio	Calcula o ruído dos atributos.
Simples	num_to_cat	Computa o número de características numéricas e categóricas.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Landmarking	one_nn_mean	Média do desempenho do classificador 1-Nearest Neighbor.
Landmarking	one_nn_mean_relative	Média relativa do desempenho do classificador 1-Nearest Neighbor.
Landmarking	one_nn_sd	Desvio padrão do desempenho do classificador 1-Nearest Neighbor.
Landmarking	one_nn_sd_relative	Desvio padrão relativo do desempenho do classificador 1-Nearest Neighbor.
Clusterização	pb	Calcular a correlação de Pearson entre correspondência de classe e distâncias de instância.
Landmarking	random_node_mean	Média do desempenho do modelo de nó de árvore de decisão única induzido por um atributo aleatório.
Landmarking	random_node_mean_relative	Média relativa do desempenho do modelo de nó de árvore de decisão única induzido por um atributo aleatório.
Landmarking	random_node_sd	Desvio padrão do desempenho do modelo de nó de árvore de decisão única induzido por um atributo aleatório.
Landmarking	random_node_sd_relative	Desvio padrão relativo do desempenho do modelo de nó de árvore de decisão única induzido por um atributo aleatório.
Estatística	range_mean	Média do calcula o intervalo (máx - min) de cada atributo.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Estatística	range_sd	Desvio padrão do calcula o intervalo (máx - min) de cada atributo.
Clusterização	sc	Calcula o número de clusters com tamanho menor que um determinado tamanho.
Estatística	sd_mean	Média do calculo do desvio padrão de cada atributo.
Estatística	sd_sd	Desvio padrão do calculo do desvio padrão de cada atributo.
Estatística	sd_ratio	Calcula um teste estatístico para homogeneidade de covariâncias.
Clusterização	sil	Calcula o valor médio da silhueta.
Estatística	skewness_mean	Calcula a média do skewness de cada atributo.
Estatística	skewness_sd	Calcula o desvio padrão do skewness de cada atributo.
Estatística	sparsity_mean	Calcula a média da métrica de esparsidade para cada atributo.
Estatística	sparsity_sd	Calcula o desvio padrão da métrica de esparsidade para cada atributo.
Estatística	t_mean_mean	Computa a média trimmed de cada atributo.
Estatística	t_mean_sd	Computa o desvio padrão trimmed de cada atributo.
Baseada em Modelo	tree_depth_mean	Calcula a média da profundidade de cada nó no modelo DT.
Baseada em Modelo	tree_depth_sd	Calcula o desvio padrão da profundidade de cada nó no modelo DT.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	**Nome**	**Descrição**
Baseada em Modelo	tree_imbalance_mean	Calcula a média do desequilíbrio de árvore para cada nó de folha.
Baseada em Modelo	tree_imbalance_sd	Calcula o desvio padrão do desequilíbrio de árvore para cada nó de folha.
Baseada em Modelo	tree_shape_mean	Calcula a média da forma de árvore para cada nó de folha.
Baseada em Modelo	tree_shape_sd	Calcula o desvio padrão da forma de árvore para cada nó de folha.
Estatística	var_mean	Calcula a média da variância de cada atributo.
Estatística	var_sd	Calcula o desvio padrão da variância de cada atributo.
Baseada em Modelo	var_importance_mean	Calcula a média da importância das características do modelo DT para cada atributo.
Baseada em Modelo	var_importance_sd	Calcula o desvio padrão da importância das características do modelo DT para cada atributo.
Clusterização	vdb	Calcula o índice de Davies e Bouldin.
Clusterização	vdu	Calcula o Índice Dunn.
Estatística	w_lambda	Calcula o valor Lambda do Wilks.
Landmarking	worst_node_mean	Desempenho médio do modelo de nó de árvore de decisão única induzido pelo pior atributo informativo.
Landmarking	worst_node_mean_relative	Desempenho médio relativo do modelo de nó de árvore de decisão única induzido pelo pior atributo informativo.

Tabela 2.1: Descrição das meta-características separadas por grupo [1]

Grupo	Nome	Descrição
Landmarking	worst_node_sd	Desvio padrão do desempenho do modelo de nó de árvore de decisão única induzido pelo pior atributo informativo.
Landmarking	worst_node_sd_relative	Desvio padrão relativo do desempenho do modelo de nó de árvore de decisão única induzido pelo pior atributo informativo.

2.3.2 Algoritmos Base e Avaliação de Desempenho

O espaço de algoritmos A é o conjunto de algoritmos candidatos no processo de recomendação. O desempenho de cada algoritmo é avaliado de acordo com uma medida selecionada, portanto, as medidas de avaliação de desempenho devem ser adequadas ao domínio do problema. Embora os algoritmos sejam tipicamente apenas classificadores ou apenas regressores, eles devem ter uma diversidade de vieses. No contexto desse trabalho, o algoritmo base pode ser definido como um *pipeline*, ou sequência de técnicas, composto de técnicas de pré-processamento e um estimador. A etapa de pré-processamento de um *pipeline* em AM é necessária e normalmente muito custosa em termos de tempo. Tipicamente, o pré-processamento de dados leva mais de 40% do tempo gasto em um tarefa de AM [57].

Embora os pesquisadores possam investigar uma gama diversificada de problemas, atualmente, devido ao armazenamento e poder computacional, eles ainda precisam transformar os dados brutos em formatos estruturados, matematicamente viáveis e computacionalmente adequados [27]. O desafio aumenta ao considerar que diferentes fontes de dados são heterogêneas e têm suas peculiaridades.

Muitos algoritmos foram desenvolvidos para a resolução de problemas no nível de pré-processamento. Geração de instâncias, discretização, tratamento de dados desbalanceados, seleção de características e filtros de ruído são exemplos de abordagens comumente aplicadas durante o pré-processamento [58]. A seleção de mais do que um pequeno grupo de abordagens de cada categoria para montar um *pipeline* pode aumentar exponencialmente o espaço de busca e a complexidade do espaço do algoritmo. Portanto, neste trabalho, o foco está nos filtros de ruído e no tratamento de dados desbalanceados.

Filtros de Ruído

Em alguns casos, rótulos ou classificações errôneas podem ser atribuídos a instâncias, resultando assim em ruído de rótulos, também chamado de ruído de classe. Outras vezes, o conjunto de dados pode não ter qualidade como resultado, por exemplo, de ruídos de sensoriamento ou de erros manuais cometidos na coleta de dados, o que gera ruído de atributo [59, 60]. Neste trabalho, foi considerado apenas o problema do ruído de classe, o tipo de ruído mais estudado.

Os filtros de ruído de classe são algoritmos para detectar e remover ruído de rótulo pelo uso de diferentes abordagens como informações de densidade, descritores de dados ou conjuntos de classificadores. O algoritmo *Edited Nearest Neighbours* (ENN) [61] é um exemplo de um filtro baseado em similaridade ou informação de densidade. Outro método comumente utilizado, o filtro *High Agreement Random Forest* (HARF) [62], utiliza o classificador *Random Forest* (RF) para identificar instâncias de ruído.

Algoritmos para Dados Desbalanceados

O desbalanceamento de classes de dados é um problema bastante comum. Ele ocorre quando uma ou mais classes são menos frequentes no conjunto de dados do que outras classes em uma tarefa de classificação [63]. Para equiparar a razão de representação das classes, e, assim, melhorar o desempenho esperado dos classificadores na classe minoritária, uma abordagem é realizar uma nova amostragem do conjunto de dados. Isto pode ser feito de duas maneiras: ou por meio de uma sobre-amostragem da classe minoritária, por exemplo, a *Synthetic Minority Over-sampling* (SMOTE)[64], ou por meio da sub-amostragem da classe majoritária, por exemplo, a *Random Undersampling* (RU) [65].

Avaliação de Classificadores e Modelos

Uma das principais dificuldades na área de Aprendizado de Máquina (AM) é identificar o melhor algoritmo de aprendizagem para a tarefa a ser resolvida. Por vezes, um classificador pode possuir melhor desempenho em uma métrica e não conseguir bons resultados de acordo com outra métrica. Desse modo, pode-se escolher aquele que possuir a característica mais adequada ao contexto, por exemplo, um classificador com tempo de resposta menor para predição, ainda que não possua uma boa acurácia [66].

Além do problema comparativo entre classificadores, por vezes é necessário identificar se o modelo treinado consegue predição satisfatória. Em alguns domínios, há ocorrência de eventos raros, o que pode comprometer não somente o treinamento, como também a avaliação do modelo. Nesse sentido, duas métricas são cruciais, o *recall* e a precisão.

Medidas derivadas dessas métricas de avaliação podem contribuir para a caracterização dos modelos obtidos [67].

A estabilidade é uma importante característica para algoritmos de AM. Um algoritmo pode ser considerado estável quando gera resultados muito próximos quando treinados com dois conjuntos de dados distintos apenas por um exemplo. Uma baixa variância pode ser considerada como uma alta estabilidade. Essa característica é importante especialmente quando o algoritmo será aplicado em conjunto de dados ruidosos [68, 69].

Para escolha e decisão do algoritmo de aprendizagem mais adequado para a resolução de um problema, são necessárias medidas de análise e comparação. Classificadores podem ser avaliados por diferentes métricas de desempenho. Todavia, ainda não há consenso sobre qual a melhor métrica para caracterizar um classificador [70]. Para melhor caracterizar um classificador, é ideal a escolha de métricas as quais capturem independentemente diferentes aspectos. Contudo, muitas métricas são correlacionadas [66].

Acurácia

Os problemas de classificação podem ser binários. Nesse tipo de problema, há apenas duas classes. Comumente os resultados das predições nesse contexto são divididos em verdadeiros positivos (tp), verdadeiros negativos (tn), falsos positivos (fp) e falsos negativos (fn). Essa medida pode dar uma ideia do desempenho do desempenho geral do classificador (Equação 2.1). Além disso, em uma matriz de confusão, pode ficar claro qual a classe com maior número de acertos e erros [71].

$$Acurácia = \frac{tp + tn}{tp + fn + fp + tn} \tag{2.1}$$

Contudo, essa medida pode ser generalizada para uso em problemas multi-classe. Todas as classes podem ser consideradas de maneira equivalente. Dessa forma, é calculada a acurácia média para todas as l classes (Equação 2.2) [71].

$$AcuráciaMédia = \sum_{i=1}^{l} \frac{\frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i}}{l} \tag{2.2}$$

Contudo, para problemas não balanceados, a acurácia pode não refletir bem o desempenho. É possível obter altos valores de acurácia ainda que um modelo apenas consiga predizer a classe majoritária. Dessa forma, outras métricas podem ser mais adequadas.

Acurácia Balanceada

Embora a acurácia seja uma medida amplamente utilizada de desempenho de um classificador, ela não é adequada para avaliar conjuntos de dados desequilibrados. A métrica

de precisão preditiva é tendenciosa para a classe majoritária [72]. Assim, a medida de acurácia balanceada pode ser usada para melhorar a avaliação em tais casos [73].

A acurácia balanceada pode ser utilizada tanto para problemas de classificação binários (Equação 2.3), quanto para multi-classe. Para o caso de problemas binários, a acurácia balanceada é calculada pela média da sensitividade somada com a especificidade [74].

$$AcuráciaBalanceada = \frac{1}{2}(\frac{tp}{tp+fn} + \frac{tn}{tn+fp}) \tag{2.3}$$

Para o caso de problemas multi-classe, assim como na acurácia padrão, é realizada a média da acurácia balanceada por todas as classes.

Capítulo 3

Metodologia Experimental

Quando MtL é utilizado, a qualidade do conjunto de meta-características escolhidas para definição de um conjunto de dados é chave na predição do desempenho de um *pipeline* de AM. Além disso, o custo de computar as meta-características não deve prolongar em demasia o treinamento e o tempo de pontuação exigidos pelos classificadores. No contexto do AutoML, o objetivo é desenvolver um sistema robusto de recomendação de *pipelines* para diferentes problemas de classificação, baseado em cálculos de caracterização reprodutíveis e viáveis.

O sistema apresenta duas fases, a fase *off-line* e a fase *on-line*. Na fase *off-line*, a meta-base, uma coleção de meta-características e acurácia balanceada de cada combinação de técnicas de pré-processamentos e classificadores para cada *dataset*, é criada e pode ser aumentada. A meta-base é parte essencial como meta-conhecimento para as predições do sistema na fase *on-line*. As análises de meta-base, nível meta e nível base são feitas para averiguar a robustez do sistema na fase *off-line*. Nessa etapa, o melhor meta-regressor para a meta-base é identificado e utilizado para as recomendações na fase *on-line*. Na fase *on-line*, as predições para novos problemas ou novos conjuntos de dados são realizadas. Nessa fase do sistema de recomendação de *pipelines*, o primeiro passo é realizar a extração das meta-características. Em seguida, as meta-características são utilizadas como entrada para os meta-regressores para predição da acurácia balanceada de cada uma das combinações. Por fim, essas informações são úteis para indicar um *pipeline* adequado.

3.1 Fase *Off-Line*

Neste trabalho, para cada conjunto de dados escolhido para compor a meta-base, foram realizadas três etapas principais. Inicialmente, foram extraídas as meta-características. Em seguida, o conjunto de dados foi pré-processado. Na última etapa, foi utilizada uma validação cruzada para obtenção da média da acurácia balanceada das combinações de

técnicas de pré-processamento e classificadores. Estas medidas coletadas são os rótulos dos meta-conjuntos de dados que formam a meta-base. Com isto, os meta-modelos foram treinados e avaliados na tarefa de predição (Figura 3.1). Também é disponibilizada a implementação do sistema de recomendação de *pipelines* [1].

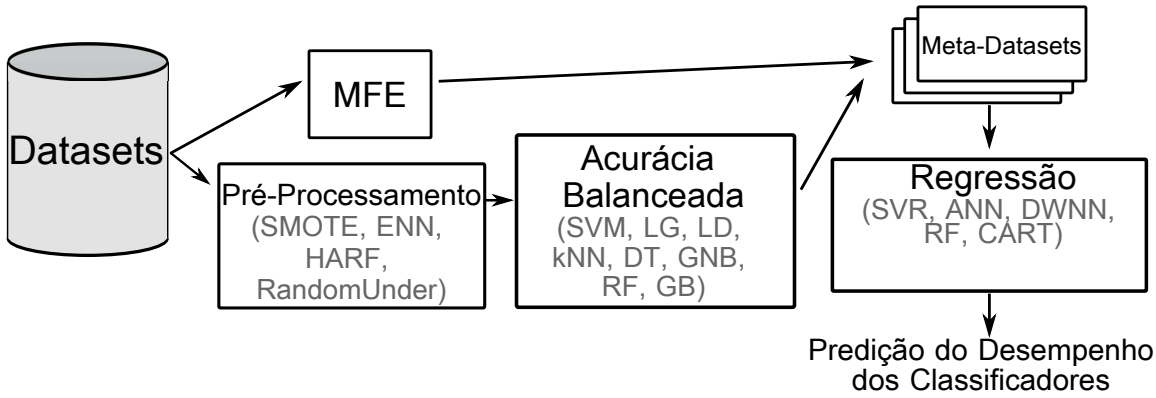


Figura 3.1: Passos para a construção da meta-base e avaliação dos meta-regressores na construção do arcabouço para o sistema de recomendação de *pipeline* baseado MtL. O conjunto de *meta-datasets* obtido forma a meta-base.

Um total de 130 meta-características diferentes foram selecionadas para definir 391 conjuntos de dados de classificação diversos. O pacote *pymfe* [1] foi utilizado para extrair as meta-características. Foram utilizados hiper-parâmetros padrão para cálculo de todas as meta-características no *pymfe*, as quais foram resumidos por média e desvio padrão. Os 391 conjuntos de dados foram obtidos do repositório OpenML [2] [52]. Foi considerada uma ampla gama de diferentes áreas, contextos e campos de estudo. Cada conjunto de dados contém um máximo de 10 classes, 10.000 amostras e 500 atributos. Todos os conjuntos de dados não possuem valores ausentes nos campos de atributos e classe.

Quatro técnicas de pré-processamento foram escolhidas: dois filtros de detecção de ruído, *Edited Nearest Neighbours* (ENN) e *High Agreement Random Forest* (HARF), e duas técnicas de re-amostragem para dados desbalanceados, *Random Undersampling* (RU) e *Synthetic Minority Over-sampling* (SMOTE). Essas técnicas foram escolhidas a fim de abranger diferentes vieses e, assim, aumentar a diversidade. É importante destacar que, para comparar a eficácia dos pré-processadores, o desempenho de predição de cada classificador também foi computado em conjuntos de dados sem a aplicação de métodos de pré-processamento, já que *pipelines* sem etapas pré-processamento também são possíveis recomendações.

Além das técnicas de pré-processamento, o sistema proposto também recomenda um algoritmo de classificação. O conjunto de algoritmos de classificação escolhido nessa im-

[1] `https://github.com/jhosoume/pipeline_recommendation`
[2] `https://www.openml.org/`

plementação é composto por *Gradient Boosting* (GB) [75], *Random Forest* (RF) [76, 77], GNB [78], Árvore de Decisão ou *Decision Tree* (DT) [79], *k-Nearest Neighbours* (k-NN) [34],*Linear Discriminant* (LD) [80], *Logistic Regression* (LR) [81], *Support Vector Machine* (SVM) [82]. Todos os classificadores usaram os parâmetros padrão apresentados no módulo Scikit-learn [83].

Nesse cotexto, há um total de 40 combinações simples possíveis de técnicas de pré-processamento e classificadores. Como apresentado, foram escolhidas quatro técnicas de pré-processamento, no entanto a ausência de aplicação de técnicas também é considerada, portanto são possíveis cinco escolhas. Já para classificadores, pode ser escolhido um classificador entre os oito classificadores apresentados. As combinações são consideradas *pipelines* simples. O aprendizado gerado por esses *pipelines* simples é utilizado para a construção de *pipelines* complexos, com múltiplas técnicas de pré-processamento.

Após a coleta de dados para construir a meta-base, cinco diferentes tipos de regressores foram escolhidos para predição da acurácia balanceada esperada de cada combinação de pré-processadores e classificadores como uma função das 130 meta-características computadas, no nível de MtL. Para recomendar o melhor *pipeline* para um determinado conjunto de dados, o sistema escolhe a combinação com o maior valor em desempenho estimado. O conjunto de cinco regressores foi escolhido para ser representativo de diferentes vieses: *Artificial Neural Network* (ANN), *Classification and Regression Tree (Decision Tree)* (CART), *Distance Weighted k-Nearest Neighbour* (DWNN), *Random Forest Regressor* (RFR) e *Support Vector Regressor* (SVR).

Para avaliar o desempenho dos regressores, foram consideradas duas análises distintas, a análise em nível meta e a análise em nível base. A análise em nível meta consiste em avaliar a competência dos meta-regressores para predição do desempenho de *pipelines*. Da mesma forma, a análise de nível base avalia as diferenças de acertos no uso dos classificadores recomendados. Na análise em nível meta, foram medidos os erros dos regressores ao estimar o desempenho preditivo de cada combinação possível por meio de *Root Mean Square Error* (RMSE). Na análise de nível base, foram avaliados os ganhos gerais da utilização da combinação recomendada em comparação com duas *baselines*: uma escolha aleatória de combinação (*Random Baseline* ou *Baseline* Aleatório (RD)) e a escolha da combinação com a maior precisão balanceada no conjunto de treinamento (*Default Baseline* ou *Baseline* de Melhor Desempenho Médio (DF)). Tanto a análise em nível meta como a análise em nível base foram avaliadas usando uma 10-*fold cross-validation*.

Após a consolidação da meta-base e treinamento dos regressores, o sistema de recomendação pode ser atualizado no modo *off-line* por novos conjuntos de dados. Para isso, as meta-características desses dados são extraídas. Em seguida, são calculadas as medidas de desempenho de cada combinação de técnica de pré-processamento e classificador. Por

fim, os regressores podem ser atualizados com as novas informações.

3.2 Fase *On-Line*

Após a construção da meta-base, treinamento e análise dos meta-modelos, o sistema pode então recomendar *pipelines* para novos conjuntos de dados. Na fase *on-line*, à medida que chegam novos conjuntos de dados, são extraídas suas meta-características, assim como na fase *off-line*. Em seguida, esses dados são usados pelos meta-modelos para predizer o desempenho preditivo das combinações em um processo iterativo. Nesse processo, o *pipeline* é formado. Em cada iteração, a melhor combinação indicada pelos meta-modelos é escolhida. Caso seja recomendada uma técnica de pré-processamento, esta é aplicada ao conjunto de dados e inserida entre os componentes do *pipeline*. Esses passos se repetem até o tamanho máximo definido para o *pipeline* ou que nenhuma técnica de pré-processamento seja recomendada (Figura 3.2).

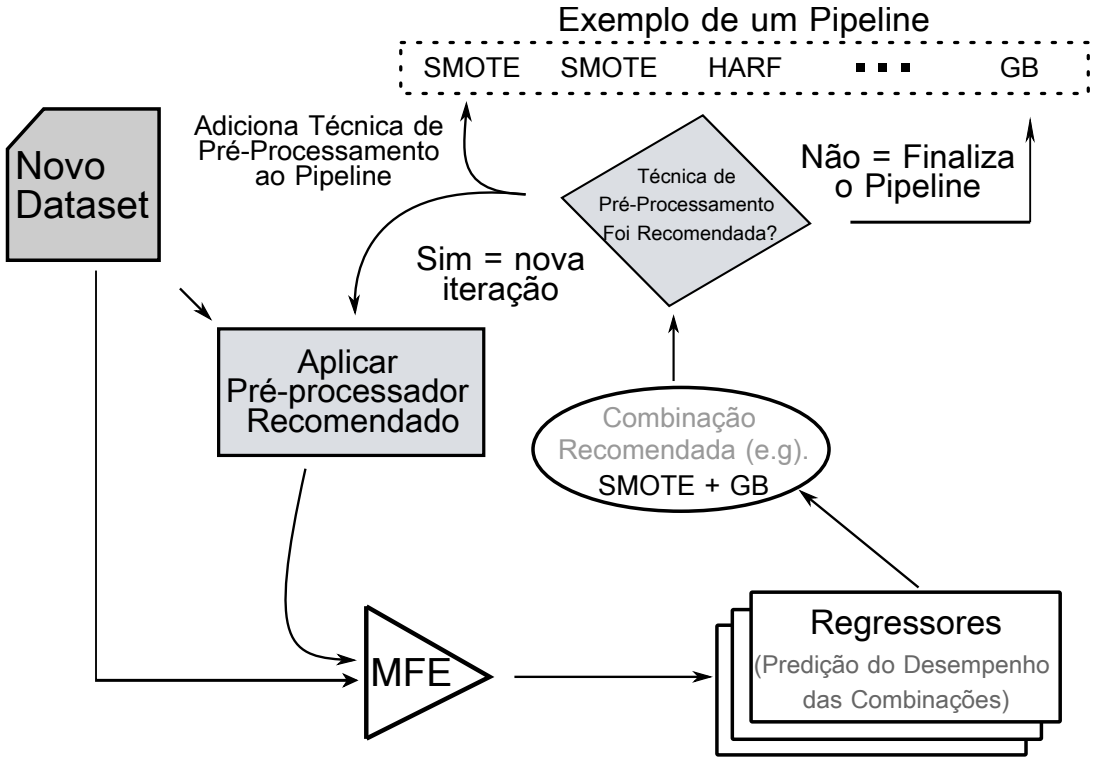


Figura 3.2: Passos para recomendação de *pipeline* baseado MtL para um novo conjunto de dados. Cada iteração consiste na repetição dos processos que compõe a fase *on-line*.

Para investigar melhor a capacidade do meta-modelo de recomendar um *pipeline* adequado, foi realizada uma avaliação da construção desse *pipeline*. O meta-regressor com o melhor resultado na análise base foi selecionado e treinado com 90% da meta-base, os

10% restantes foram usados para testes. Em cada iteração, são avaliadas as predições. Uma predição é considerada correta se é igual à combinação da máxima precisão balanceada para um dado conjunto de dados. Então, a soma de acertos para todas as iterações é computada. Cada iteração corresponde à aplicação de um pré-processador a um conjunto de dados. Se o meta-regressor não prevê corretamente o pré-processador ou indica que não é necessário o pré-processamento, a avaliação é interrompida e a avaliação do conjunto de dados não avança para a próxima rodada. Devido ao custo computacional, limitamos o número de técnicas de pré-processamento a serem aplicadas sequencialmente a um conjunto de dados a um máximo de 5 iterações.

Posteriormente, para cada conjunto de dados de teste, foi utilizado o sistema proposto para recomendar, em cada etapa de construção do *pipeline*, uma combinação. Se a combinação recomendada for composta por uma técnica de pré-processamento, a técnica é adicionada ao *pipeline* recomendado. Assim, a técnica de pré-processamento é aplicada ao conjunto de dados e as meta-características são extraídas novamente. Caso contrário, o algoritmo de classificação é adicionado ao *pipeline* e o sistema retorna o *pipeline* como uma recomendação. O processo continua até que uma das três condições indique o ponto de parada. Estas condições são: nenhuma técnica de pré-processamento é recomendada, o número de técnicas de pré-processamento atinge o limite de iterações, ou a técnica recomendada é diferente da melhor combinação previamente medida na iteração. O mesmo procedimento foi aplicado ao DF, no qual, para todos os conjuntos de dados e todas as rodadas, indica a combinação com o máximo desempenho médio na meta-base. Esta experiência foi realizada 10 vezes independentes.

3.3 *Hardware*

Todos os cálculos foram executados em um nó de cluster com dois processadores Intel Xeon Silver 4114 de 2,20GHz com 256 Gb de RAM. O sistema operacional instalado é o Debian OS versão 9.

3.4 *Software*

As linguagens de programação usadas para implementar os experimentos foram Python 3 e R. O pacote principal usado foi o Scikit-learn[3]. Para armazenar as informações obtidas foi utilizado MySQL[4]. O desenho do banco de dados está apresentado na Figura 3.3.

[3] `https://scikit-learn.org/stable/`
[4] `https://www.mysql.com/`

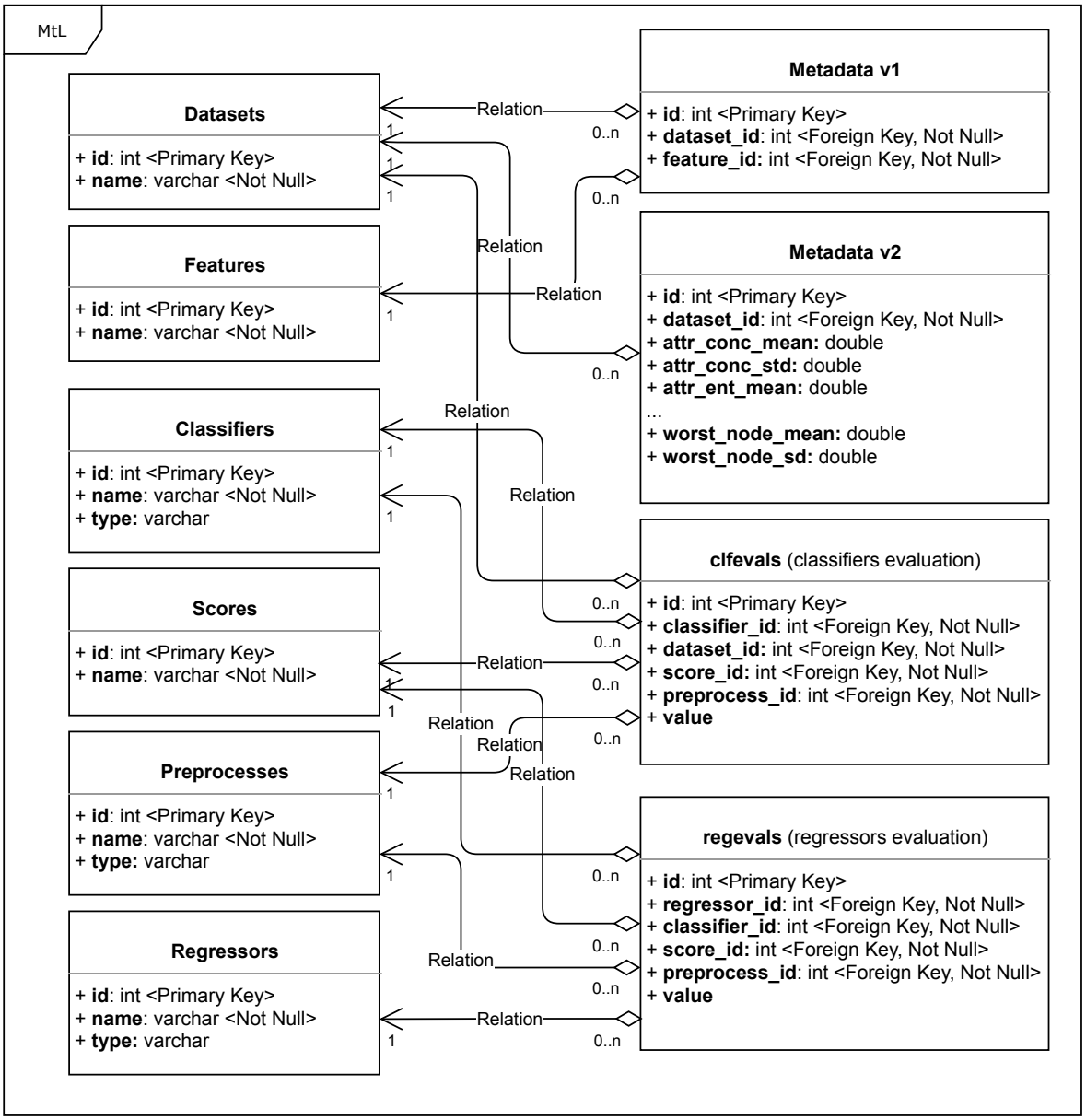


Figura 3.3: Esquema das tabelas e seus atributos para montagem de um sistema de recomendação utilizando Meta-Aprendizado ou *Meta-Learning* (MtL).

Visto que os algoritmos de filtro de ruído ainda não haviam sido implementados em Python, foi feita a implementação de alguns dos algoritmos presentes no pacote *NoiseFiltersR*[5]. Esse módulo foi disponibilizado no *GitHub*[6]. O sistema de recomendação proposto está sendo contruído em um sistema maior de AutoML [7].

[5] `https://cran.r-project.org/web/packages/NoiseFiltersR/index.html`
[6] `https://github.com/jhosoume/NoiseFiltersPy`
[7] `https://github.com/jhosoume/curumin_mtl`

Capítulo 4

Resultados

Neste capítulo, é analisada a recomendação proposta de uma *pipeline* que consiste na aplicação de técnicas de pré-processamento em um determinado conjunto de dados e um classificador adequado. São apresentados os resultados experimentais para as quatro análises, meta-base, meta-nível, nível de base e avaliação dos *pipelines*. Na análise da meta-base, é mostrado o desempenho de cada par de combinações de uma técnica de pré-processamento e um algoritmo de classificação, em média e número de acertos em comparação com os demais. Na análise em meta-base, são descritas as medidas de desempenho dos regressores na tarefa de prever a precisão balanceada de cada possível combinação de pares. Na análise de nível de base, o sistema de recomendação proposto é comparado com os *baselines* RD e DF. Também é apresentada uma análise da importância das meta-características em relação aos modelos RFR. No estudo da construção de *pipelines*, são analisadas as técnicas previstas corretamente em cada iteração de construção do *pipeline* de acordo com um desempenho da combinação previamente medido.

4.1 Análise da Meta-Base

A meta-base é composta pelo desempenho de um conjunto composto de combinação de técnicas de pré-processamento e um classificador base. Nela, estão contidas as meta-características selecionadas de 391 conjuntos de dados de tarefas binárias e multi-classe. Cada conjunto de dados é submetido a quatro técnicas de pré-processamento diferentes, HARF, ENN, RU e SMOTE, e depois submetido à avaliação de classificação.

Em detalhes, na Figura 4.1, é apresentada a média da acurácia para cada combinação de pré-processador e classificador. Quanto mais escura a cor, melhor aquela combinação aplicada nos conjuntos de dados em média. A combinação de SMOTE e RU com GB alcançou os resultados mais promissores. Enquanto isso, o SVM combinado com qualquer técnica de pré-processamento obteve os piores resultados, especialmente na au-

sência de pré-processamento ou quando ENN é utilizado. No entanto, uma vez que todas as combinações atingiram valores de 60% de acurácia balanceada em média, elas foram consideradas adequadas para as análises a seguir.

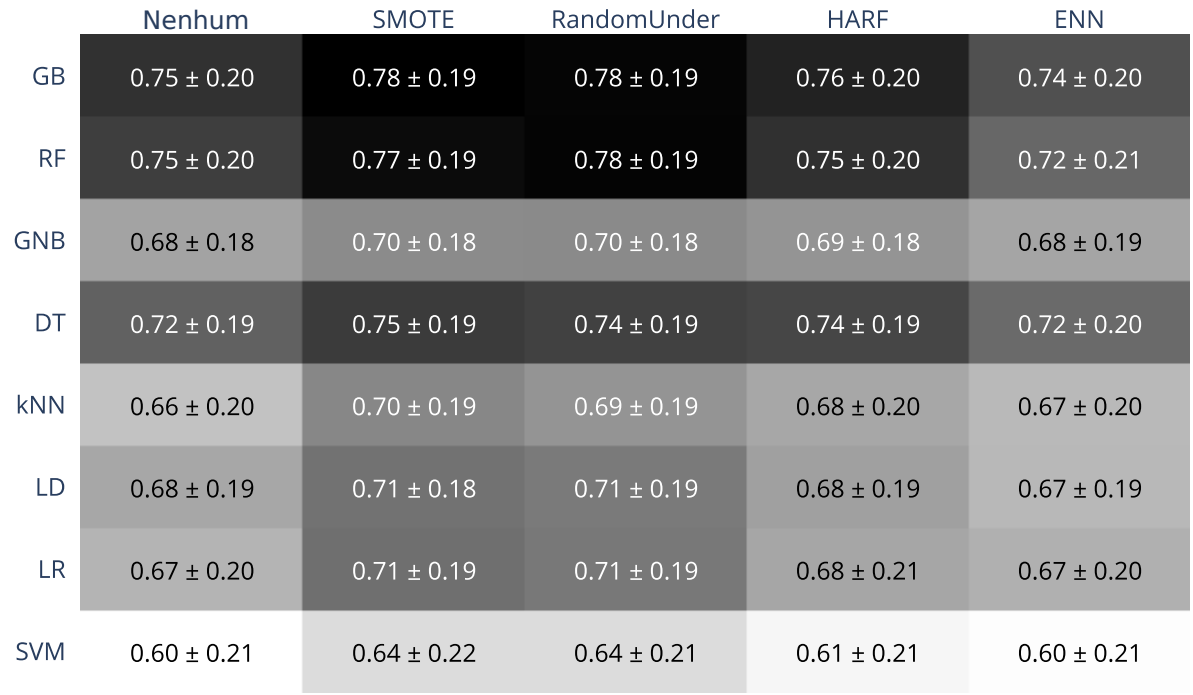


Figura 4.1: Análise da meta-base. Média e desvio padrão de acurácia balanceada para uma combinação de um pré-processador e um algoritmo de classificação. No eixo x estão os pré-processadores e no eixo y estão os classificadores.

O desempenho médio das técnicas não é suficiente para justificar a recomendação de uma combinação. Deve-se almejar a diversificação das combinações, o que poderia levar a uma competição entre as combinações pelo melhor desempenho, com cada uma delas tendo um desempenho melhor que as outras em, pelo menos, um pequeno subconjunto de conjuntos de dados. Mostra-se aqui que, embora algumas combinações tenham melhor desempenho em geral, outras combinações ainda têm melhor desempenho em um subconjunto do conjunto de dados.

Com o objetivo de melhor representar a meta-base, a distribuição das combinações vencedoras, em outras palavras, o pré-processador e classificador com a maior acurácia balanceada para cada conjunto de dados, é ilustrada na Figura 4.2. Dessa forma, uma vitória ou acerto para uma combinação é a obtenção do melhor resultado de uma métrica para um conjunto de dados apresentado dentre todas as combinações. Quando mais de uma combinação apresentou o maior valor de acurácia balanceada, uma vitória é adicionado a ambos. Similar aos resultados na Figura 4.1, GB foi o classificador vencedor, com um total de 135 vitórias. Com relação às técnicas de pré-processamento, SMOTE

atingiu os melhores resultados, totalizando, com um total de 202 vitórias. Neste cenário, cada combinação de classificador e pré-processamento tem pelo menos cinco conjuntos de dados vencedores, nos quais tem a maior acurácia balanceada. No entanto, a meta-base é desbalanceada por classe, já que as combinações que envolvem GB, RF e SMOTE concentram a maior parte dos exemplos vencedores.

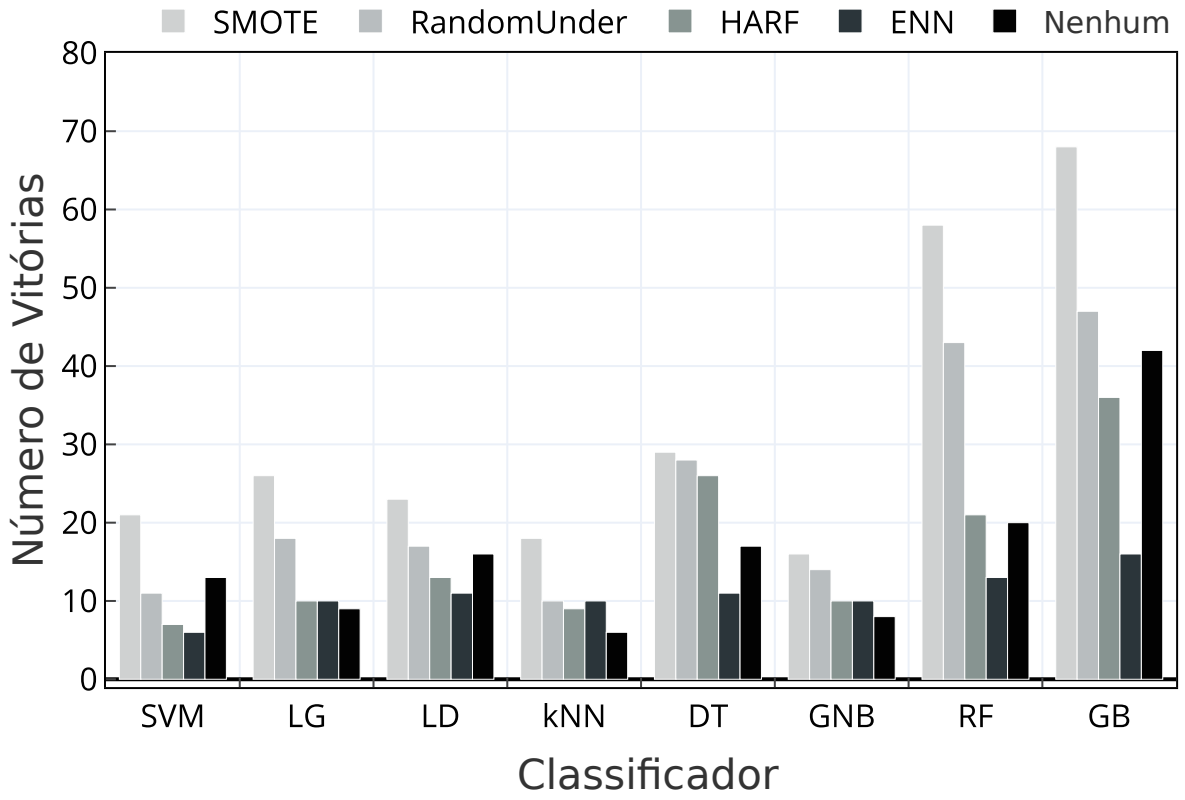


Figura 4.2: Distribuição das vitórias para cada combinação de uma técnica de pré-processamento e um algoritmo de classificação. Cada vitória de uma combinação é dada quando essa combinação obtém o maior valor de acurácia balanceada para um conjunto de dados dentre todas as combinações consideradas.

4.2 Análise no Nível Meta

Com base nos resultados obtidos da análise da meta-base (Seção 4.1), a recomendação de um *pipeline* foi considerada um problema de regressão. Neste sentido, ANN, CART, DWNN, SVR e RFR foram escolhidos para prever a acurácia balanceada esperada para uma combinação com base nas meta-características dos conjuntos de dados. Após o treinamento adequado dos meta-regressores, cada um deles teve seu desempenho avaliado. Portanto, um total de 200 meta-regressores foram avaliados, cinco tipos diferentes de

regressores para quatro pré-processadores e oito classificadores. Estes são necessários para cada iteração na metodologia.

A Figura 4.3 mostra o *Root Mean Square Error* (RMSE) de cada regressor em *boxplots*, separados por técnicas de pré-processamento. Cada gráfico agrupa valores obtidos para um pré-processador e cada *boxplot* resume os resultados estimados de um meta-regressor (em branco) sobre todos os classificadores base. O RMSE foi calculado por um processo de 10-*fold cross-validation*. Os dois últimos *boxplots* (em cinza) de cada gráfico, RD e DF, são as *baselines* com as quais os meta-regressores são comparados. A DF é a estratégia de retornar a acurácia balanceada média para qualquer predição, enquanto a RD reporta um número aleatório entre o valor mínimo e máximo de todos os valores de acurácia balanceada observados.

Usando o teste estatístico *Friedman* com o pós-teste *Nemenyi* com intervalo de confiança de 95% [84], o CART, DWNN e RFR apresentaram melhor desempenho preditivo do que as linhas de base, o que indica que eles predizem o desempenho das combinações com mais precisão. No entanto, a ANN é tão propensa a erros quanto os dois *baselines*. O RFR teve melhor desempenho do que os outros meta-regressores. Não há diferença significativa no desempenho considerando os pré-processadores.

4.3 Análise no Nível Base

Para adotar um regressor adequado para o sistema de recomendação, as propostas em média devem superar a estratégia de selecionar um *pipeline* aleatório (RD) ou um *pipeline* de melhor média (DF) [24]. A seguir, são avaliadas as recomendações dadas pelo sistema baseado em MtL. Para cada algoritmo de regressão, o sistema recomenda a combinação com o mais alto desempenho preditivo.

A Figura 4.4 mostra o ganho ou perda em porcentagem das recomendações comparativamente a cada *baseline*. A diferença de desempenho das recomendações dadas por cada regressor (y_R) e das recomendações dadas pelas *baselines* (y_B) para todos os conjuntos de dados (P, tal que $p \in P$) são somadas e, em seguida, normalizadas pela soma dos ganhos do melhor *pipeline* possível (y_{max}):

$$Ganho = \frac{\sum_p^P (y_R(p) - y_B(p))}{\sum_p^P y_{max}(p)} \tag{4.1}$$

Esta predição foi realizada dez vezes independentemente. Enquanto ANN e CART só superam a *Random Baseline* ou *Baseline* Aleatório (RD), DWNN, RFR e SVR têm melhores ganhos em acurácia balanceada do que RD e DF. Neste sentido, a coleta das meta-características de um conjunto de dados, seguida do uso posterior do sistema de

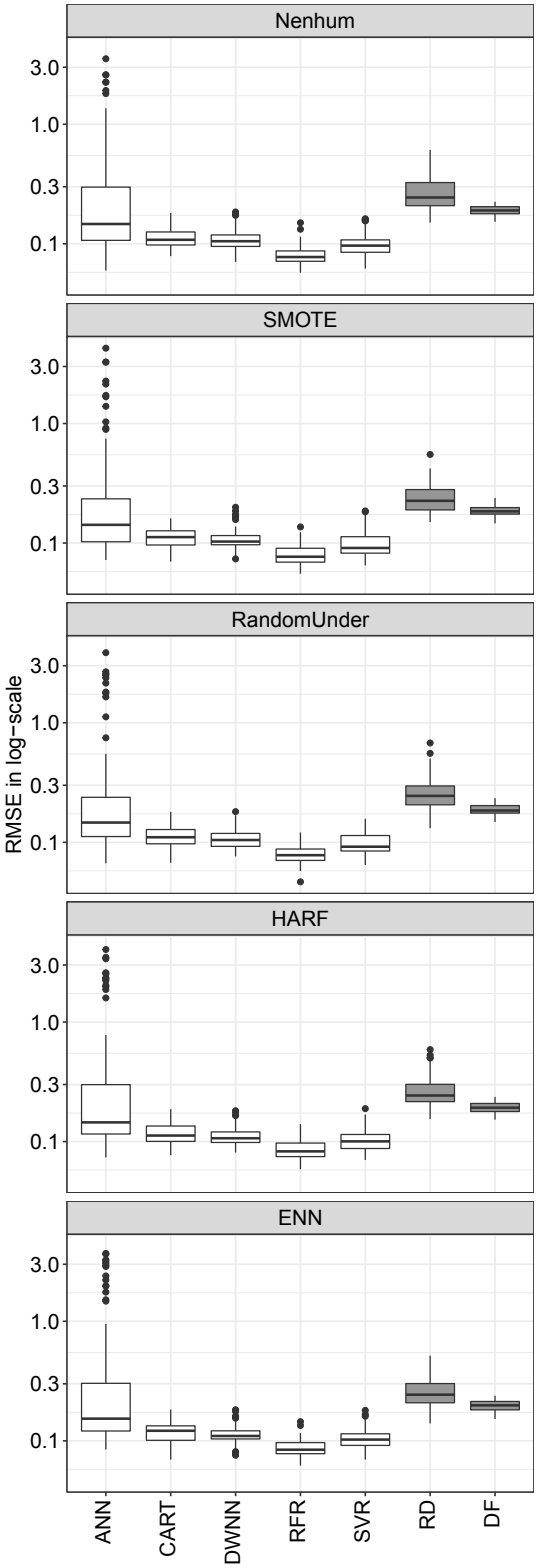


Figura 4.3: RMSE de cada meta-regressor por cada pré-processador, agrupando os resultados por classificador.

recomendação, leva a uma maior acurácia balanceada na tarefa de classificação através da aplicação da combinação recomendada. Este resultado complementam os anteriores [24], já que a recomendação não considera apenas os algoritmos de classificação, mas também o intrincado processo de escolha de uma técnica de pré-processamento adequada.

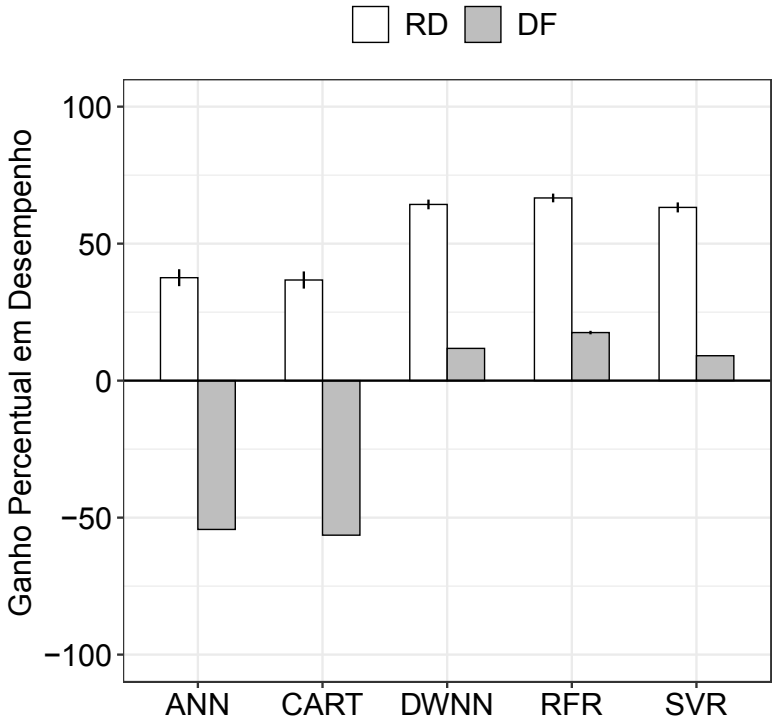


Figura 4.4: Melhoria na acurácia balanceada do classificador a partir da recomendação sobre as *baselines* quando normalizada pela melhor acurácia balanceada na meta-base.

4.4 Características Importantes do *Random Forest*

A fim de entender melhor quais meta-características contribuíram mais para resultados alcançados, foram estudadas as meta-características utilizadas pelo regressor *Random Forest Regressor* (RFR). A Figura 4.5 mostra as vinte primeiras meta-características mais importante para RFR, de acordo com sua contribuição individual para a redução do *Mean Square Error* (MSE), bem como seus correspondentes grupos de meta-características. A contribuição foi medida como o aumento do MSE quando essa meta-característica foi omitida. O eixo x lista as meta-características em ordem decrescente de relevância, com a mudança média no MSE mostrada no eixo y. As barras verticais indicam erros padrão de estimativa.

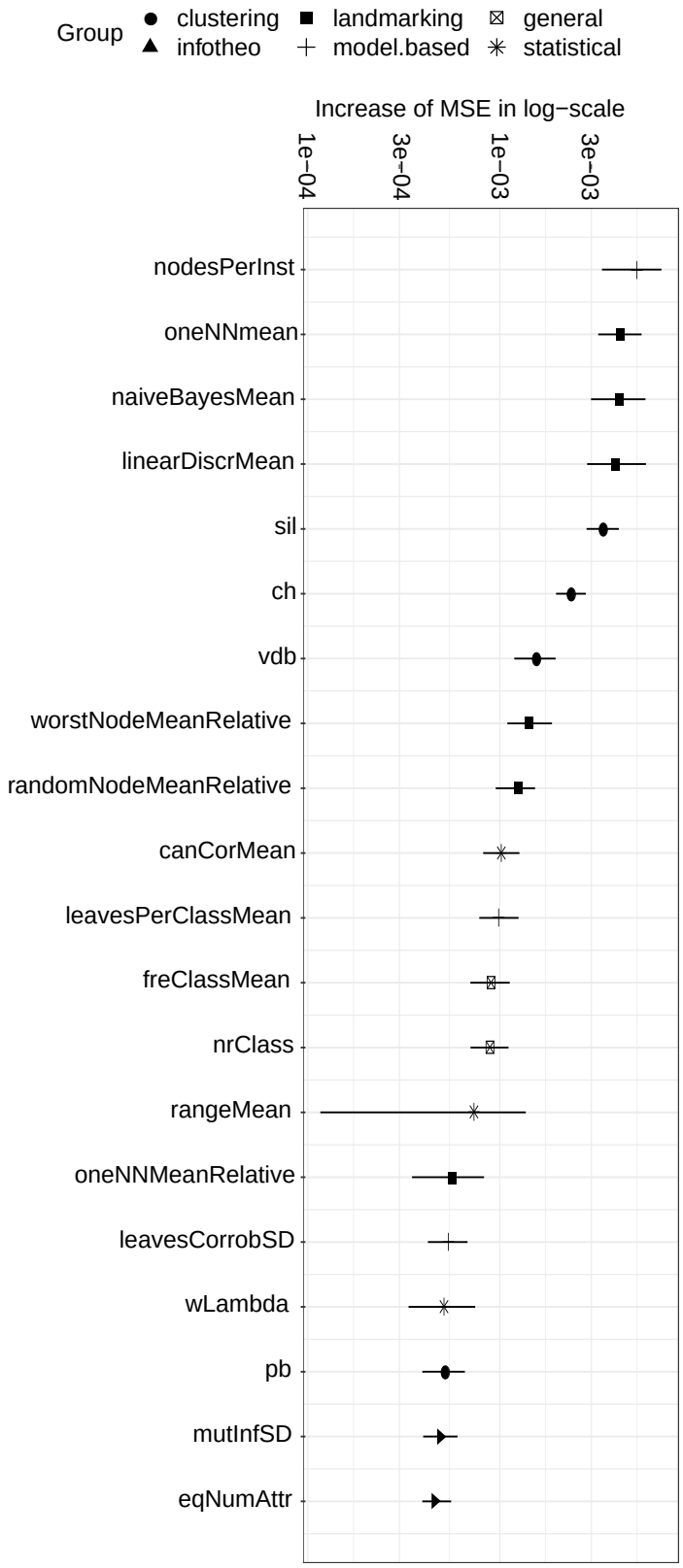


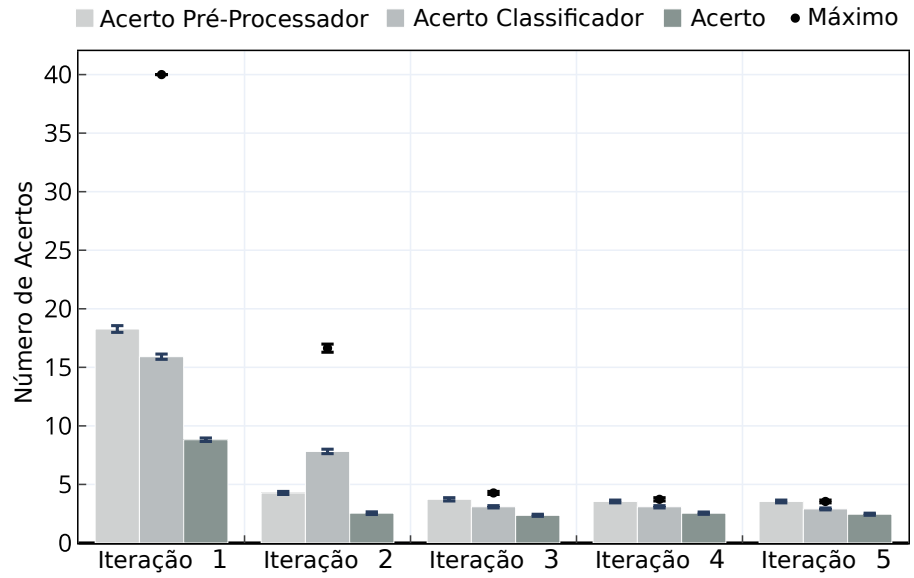
Figura 4.5: Impacto das meta-características sobre a capacidade de predição do *Random Forest Regressor* (RFR) indicada pelo *Root Mean Square Error* (RMSE).

A meta-característica com maior impacto na predição foi a mediada de caracterização baseada em modelo de nós por instância. Uma medida da proporção de nós não-folha por número de instâncias em um modelo de árvore de decisão. O grupo de meta-características de *landmarking* é o mais relevante, seguido pelas medidas de agrupamento.

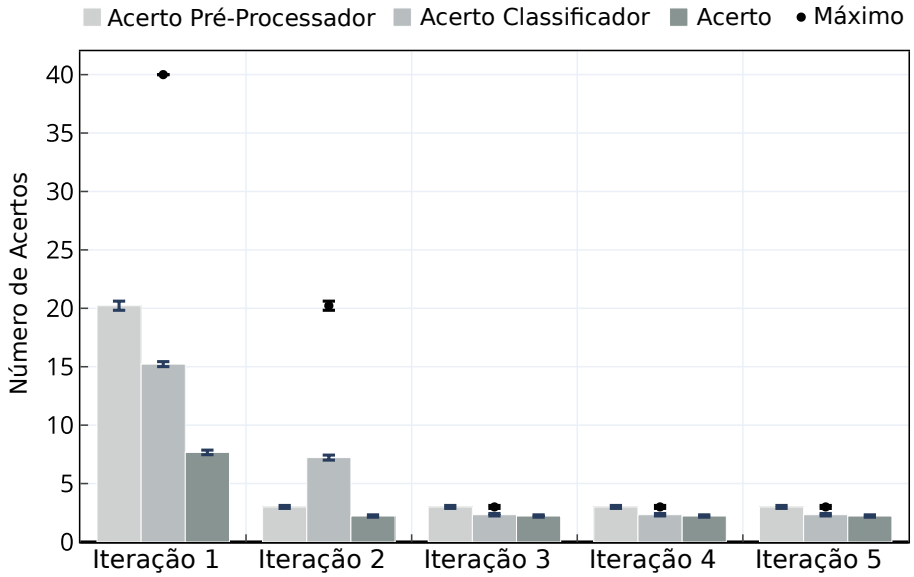
4.5 Análise da Recomendação de *Pipelines*

A eficácia do sistema de recomendação para predizer um *pipeline* de classificação adequadamente composto de múltiplas etapas de pré-processamento é mostrada na Figura 4.6. Pelo menos 40% das sugestões foram obtidas corretamente para os conjuntos de teste na primeira iteração, considerando técnicas de pré-processamento e predição de algoritmos de classificação de forma independente. Quase 25% dos *pipelines* previstos na primeira iteração corresponderam totalmente aos *pipelines* com o melhor desempenho preditivo. Quando comparado com o *baseline default*, o sistema de recomendação baseado em RF era melhor em todos os casos, exceto a recomendação de técnicas de pré-processamento na primeira rodada.

Para ambos, RF e DF, o número de conjuntos de dados avaliados diminuiu a cada iteração, uma vez que a predição dos métodos de pré-processamento não coincidia com o *pipeline* de melhor desempenho aferido, ou a recomendação indicou que não era necessário nenhum outro pré-processamento. Com este resultado, uma hipótese levantada é que não apenas a aplicação consecutiva de técnicas de pré-processamento, como filtros de ruído, poderia degenerar os conjuntos de dados ao longo das iterações, como também os conjuntos de dados que já têm seus *pipelines* corretamente previstos em estágios anteriores poderiam ser menos complexos, levando assim a um aumento da taxa de acertos em iterações posteriores.



(a) Random Forest



(b) Default

Figura 4.6: Número de acertos de combinação do meta-regressor por iteração. As barras indicam o número de predições corretas das técnicas de pré-processamento, classificadores e a combinação dos dois. Os pontos indicam o número de conjuntos de dados (máximo) avaliados por iteração.

Capítulo 5

Conclusão

Neste trabalho, um sistema baseado em MtL para recomendar uma combinação de técnica de pré-processamento com um algoritmo de classificação foi proposto e avaliado experimentalmente. A recomendação de um *pipeline* composto de múltiplas etapas de pré-processamento e um classificador foi investigado. Foi realizado um extenso e reprodutível processo de extração de meta-características para caracterizar um conjunto heterogêneo de tarefas de classificação de diferentes domínios. Neste contexto, os meta-regressores foram treinados para prever a acurácia balanceada esperada de combinações de quatro técnicas de pré-processamento e oito diferentes algoritmos de classificação.

O desempenho dos meta-regressores neste problema foram comparados com dois *baselines*, o *Default Baseline* ou *Baseline* de Melhor Desempenho Médio (DF) e o *Random Baseline* ou *Baseline* Aleatório (RD). *Random Forest Regressor* (RFR) foi o melhor meta-modelo em todos os casos e, portanto, selecionado como algoritmo de regressão para o sistema de recomendação de *pipelines*. Na tarefa de construção do *pipeline*, os modelos induzidos por RFR foram capazes de superar uma estratégia padrão, recomendando adequadamente *pipelines* complexos para aproximadamente 40% dos conjuntos de dados.

O sistema proposto, ainda que possa ser aplicado para problemas de diversos domínios, assim como para problema de regressão, possui diversas limitações. A principal delas está na necessidade de adquirir informações de desempenho para a inclusão no conjunto de recomendações. Esse passo demanda tempo computacional para o treino apropriado dos regressores utilizados. Além disso, a expansão no número de técnicas consideradas pode impactar no desempenho dos regressores adotados. Um outro ponto está na ausência de uma etapa de otimização dos hiper-parâmetros dos regressores. Como proposto, essa etapa possivelmente seria realizada externamente por outro método, como a otimização Bayesiana.

Como trabalho futuro, espera-se analisar em mais detalhes outros regressores e reduzir o número de meta-características necessárias. Além disso, o aumento do sistema para

uma abordagem de ranqueamento beneficiaria uma decisão informada sobre o *pipeline* mais adequado para uma determinada tarefa de classificação. Especificamente, planeja-se investigar o impacto do pré-processamento repetido de um conjunto de dados.

Referências

[1] Alcobaça, Edesio, Felipe Siqueira, Adriano Rivolli, Luís P. F. Garcia, Jefferson T. Oliva e André C. P. L. F. de Carvalho: *MFE: Towards reproducible meta-feature extraction.* Journal of Machine Learning Research, 21(111):1–5, 2020, ISSN 1533-7928. `http://jmlr.org/papers/v21/19-348.html`, acesso em 2020-07-09. x, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 25

[2] McCulloch, Warren S. e Walter Pitts: *A logical calculus of the ideas immanent in nervous activity.* The bulletin of mathematical biophysics, 5(4):115–133, dezembro 1943, ISSN 1522-9602. `https://doi.org/10.1007/BF02478259`, acesso em 2019-11-22. 1

[3] Solomonoff, Raymond J: *An inductive inference machine.* Em *IRE Convention Record, Section on Information Theory*, volume 2, páginas 56–62, 1957. 1

[4] Shoham, Yoav, Raymond Perrault, Erik Brynjolfsson, Jack Clark, James Manyika, Juan Carlos Niebles, Terah Lyons, John Etchemendy e Barbara Grosz: *The AI Index 2018 Annual Report.* Relatório Técnico, AI Index Steering Committee, dezembro 2018. `https://aiindex.org`, acesso em 2019-11-21. 1

[5] Magoulas, George D. e Andriana Prentza: *Machine Learning in Medical Applications.* Em Paliouras, Georgios, Vangelis Karkaletsis e Constantine D. Spyropoulos (editores): *Machine Learning and Its Applications: Advanced Lectures*, Lecture Notes in Computer Science, páginas 300–307. Springer, Berlin, Heidelberg, 2001, ISBN 978-3-540-44673-6. `https://doi.org/10.1007/3-540-44673-7_19`, acesso em 2019-11-21. 1

[6] Wei, Wei, Shyam Visweswaran e Gregory F. Cooper: *The application of naive Bayes model averaging to predict Alzheimer's disease from genome-wide data.* Journal of the American Medical Informatics Association, 18(4):370–375, julho 2011, ISSN 1067-5027. `https://academic.oup.com/jamia/article/18/4/370/732731`, acesso em 2019-11-22. 1

[7] Cruz, Joseph A. e David S. Wishart: *Applications of Machine Learning in Cancer Prediction and Prognosis.* Cancer Informatics, 2:117693510600200030, janeiro 2006, ISSN 1176-9351. `https://doi.org/10.1177/117693510600200030`, acesso em 2019-11-21. 1

[8] Galatzer-Levy, Isaac R., Karen Inge Karstoft, Alexander Statnikov e Arieh Y. Shalev: *Quantitative forecasting of PTSD from early trauma responses: A Machine Learning application.* Journal of Psychiatric Research, 59:68–76, dezembro

2014, ISSN 0022-3956. http://www.sciencedirect.com/science/article/pii/S002239561400260X, acesso em 2019-11-21. 1

[9] Rafiei, Mohammad Hossein e Hojjat Adeli: *A novel machine learning-based algorithm to detect damage in high-rise building structures*. The Structural Design of Tall and Special Buildings, 26(18):e1400, 2017, ISSN 1541-7808. https://onlinelibrary.wiley.com/doi/abs/10.1002/tal.1400, acesso em 2019-11-21. 1

[10] Zander, S., T. Nguyen e G. Armitage: *Automated traffic classification and application identification using machine learning*. Em *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, páginas 250–257, novembro 2005. ISSN: 0742-1303. 1

[11] Sinclair, C., L. Pierce e S. Matzner: *An application of machine learning to network intrusion detection*. Em *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, páginas 371–377, dezembro 1999. ISSN: 1063-9527. 1

[12] Briand, Lionel C.: *Novel Applications of Machine Learning in Software Testing*. Em *2008 The Eighth International Conference on Quality Software*, páginas 3–10, agosto 2008. ISSN: 2332-662X. 1

[13] Wolpert, David H.: *Stacked generalization*. Neural Networks, 5(2):241–259, janeiro 1992, ISSN 0893-6080. http://www.sciencedirect.com/science/article/pii/S0893608005800231, acesso em 2020-04-07. 1, 5

[14] Adam, Stavros P., Stamatios Aggelos N. Alexandropoulos, Panos M. Pardalos e Michael N. Vrahatis: *No Free Lunch Theorem: A Review*. Em Demetriou, Ioannis C. e Panos M. Pardalos (editores): *Approximation and Optimization : Algorithms, Complexity and Applications*, Springer Optimization and Its Applications, páginas 57–82. Springer International Publishing, Cham, 2019, ISBN 978-3-030-12767-1. https://doi.org/10.1007/978-3-030-12767-1_5, acesso em 2020-08-29. 1, 6

[15] Piatetsky-Shapiro, Gregory, R. Brachman, T. Khabaza, W. Klösgen e Evangelos Simoudis: *An Overview of Issues in Developing Industrial Data Mining and Knowledge Discovery Applications*. Em *Knowledge Discovery and Data Mining*, 1996. 1

[16] Guyon, Isabelle, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, Alexander Statnikov, Sébastien Treguer e Evelyne Viegas: *A Brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning Without Human Intervention*. Em *Workshop on Automatic Machine Learning*, páginas 21–30, dezembro 2016. http://proceedings.mlr.press/v64/guyon_review_2016.html, acesso em 2020-08-06, ISSN: 1938-7228 Section: Machine Learning. 2, 6

[17] Olson, Randal S. e Jason H. Moore: *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*. Em Hutter, Frank, Lars Kotthoff e Joaquin Vanschoren (editores): *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, páginas 151–160. Springer International Publishing, Cham, 2019, ISBN 978-3-030-05318-5. https://doi.org/10.1007/978-3-030-05318-5_8, acesso em 2020-08-06. 2, 6, 7

[18] Thornton, Chris, Frank Hutter, Holger H. Hoos e Kevin Leyton-Brown: *Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms*. arXiv:1208.3719 [cs], março 2013. http://arxiv.org/abs/1208.3719, acesso em 2020-08-07, arXiv: 1208.3719. 2, 7

[19] Laadan, Doron, Roman Vainshtein, Yarden Curiel, Gilad Katz e Lior Rokach: *RankML: a Meta Learning-Based Approach for Pre-Ranking Machine Learning Pipelines*. arXiv:1911.00108 [cs, stat], novembro 2019. http://arxiv.org/abs/1911.00108, acesso em 2020-04-07, arXiv: 1911.00108. 2, 6, 7

[20] Brazdil, Pavel, Christophe Giraud Carrier, Carlos Soares e Ricardo Vilalta: *Metalearning: Applications to Data Mining*. Cognitive Technologies. Springer-Verlag, Berlin Heidelberg, 2009, ISBN 978-3-540-73262-4. https://www.springer.com/gp/book/9783540732624, acesso em 2019-11-22. 2, 7

[21] Vanschoren, Joaquin, Hendrik Blockeel, Bernhard Pfahringer e Geoffrey Holmes: *Experiment databases*. Machine Learning, 87(2):127–158, maio 2012, ISSN 1573-0565. https://doi.org/10.1007/s10994-011-5277-0, acesso em 2020-04-03. 2, 5

[22] Prodromidis, Andreas, Philip Chan, Salvatore Stolfo e others: *Meta-learning in distributed data mining systems: Issues and approaches*. Advances in distributed and parallel knowledge discovery, 3:81–114, 2000. Publisher: AAAI/MIT Press Menlo Park. 2, 7

[23] Feurer, Matthias, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum e Frank Hutter: *Efficient and Robust Automated Machine Learning*. Em Cortes, C., N. D. Lawrence, D. D. Lee, M. Sugiyama e R. Garnett (editores): *Advances in Neural Information Processing Systems 28*, páginas 2962–2970. Curran Associates, Inc., 2015. http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf, acesso em 2020-08-07. 2, 7

[24] Garcia, Luís P. F., Ana C. Lorena, Marcilio C. P. de Souto e Tin Kam Ho: *Classifier Recommendation Using Data Complexity Measures*. Em *2018 24th International Conference on Pattern Recognition (ICPR)*, páginas 874–879, agosto 2018. ISSN: 1051-4651. 2, 9, 34, 36

[25] Schoenfeld, Brandon, Christophe Giraud-Carrier, Mason Poggemann, Jarom Christensen e Kevin Seppi: *Preprocessor Selection for Machine Learning Pipelines*. arXiv:1810.09942 [cs, stat], outubro 2018. http://arxiv.org/abs/1810.09942, acesso em 2020-04-07, arXiv: 1810.09942. 2

[26] Garcia, Luís P. F., André C. P. L. F. de Carvalho e Ana C. Lorena: *Noise detection in the meta-learning level*. Neurocomputing, 176:14–25, fevereiro 2016, ISSN 0925-2312. http://www.sciencedirect.com/science/article/pii/S0925231215005482, acesso em 2020-04-06. 2

[27] Bilalli, Besim, Alberto Abelló, Tomàs Aluja-Banet e Robert Wrembel: *Intelligent assistance for data pre-processing*. Computer Standards & Interfaces, 57:101–109,

março 2018, ISSN 0920-5489. http://www.sciencedirect.com/science/article/pii/S0920548916302306, acesso em 2020-08-23. 2, 20

[28] Vilalta, Ricardo, Christophe Giraud-Carrier e Pavel Brazdil: *Meta-Learning - Concepts and Techniques*. Em Maimon, Oded e Lior Rokach (editores): *Data Mining and Knowledge Discovery Handbook*, páginas 717–731. Springer US, Boston, MA, 2010, ISBN 978-0-387-09823-4. https://doi.org/10.1007/978-0-387-09823-4_36, acesso em 2019-11-23. 2

[29] Vanschoren, Joaquin: *Meta-Learning: A Survey*. arXiv:1810.03548 [cs, stat], outubro 2018. http://arxiv.org/abs/1810.03548, acesso em 2019-11-23, arXiv: 1810.03548. 2, 5

[30] Muñoz, M. A., Villanova, L, Baatar, D. e Smith-Miles, K.: *Instance spaces for machine learning classification*. Machine Learning, (107):109–147, 2018. https://link.springer.com/article/10.1007/s10994-017-5629-5, acesso em 2020-04-03. 2

[31] Rivolli, Adriano, Luís P. F. Garcia, Carlos Soares, Joaquin Vanschoren e André C. P. L. F. de Carvalho: *Characterizing classification datasets: a study of meta-features for meta-learning*. arXiv:1808.10406 [cs, stat], agosto 2019. http://arxiv.org/abs/1808.10406, acesso em 2020-07-09, arXiv: 1808.10406 version: 2. 3, 8, 9

[32] Legg, Shane e Marcus Hutter: *A Collection of Definitions of Intelligence*. Em *Proceedings of the 2007 Conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006*, páginas 17–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press, ISBN 978-1-58603-758-1. http://dl.acm.org/citation.cfm?id=1565455.1565458, acesso em 2019-11-22. 4

[33] Wang, Hua, Cuiqin Ma e Lijuan Zhou: *A Brief Review of Machine Learning and Its Application*. Em *2009 International Conference on Information Engineering and Computer Science*, páginas 1–4, dezembro 2009. ISSN: 2156-7387. 4

[34] Mitchell, Thomas M.: *Machine Learning*. McGraw-Hill, Inc., março 1997, ISBN 978-0-07-042807-2. http://dl.acm.org/citation.cfm?id=541177, acesso em 2019-11-22. 4, 5, 26

[35] Wolpert, David H.: *The Lack of A Priori Distinctions Between Learning Algorithms*. Neural Computation, 8(7):1341–1390, outubro 1996, ISSN 0899-7667. https://doi.org/10.1162/neco.1996.8.7.1341, acesso em 2019-11-22. 5

[36] Gordon, Diana F. e Marie Desjardins: *Evaluation and selection of biases in machine learning*. Machine Learning, 20(1):5–22, julho 1995, ISSN 1573-0565. https://doi.org/10.1007/BF00993472, acesso em 2019-11-22. 5

[37] Wolpert, D.H. e W.G. Macready: *No free lunch theorems for optimization*. IEEE Transactions on Evolutionary Computation, 1(1):67–82, abril 1997, ISSN 1941-0026. 5, 6

[38] Schaffer, Cullen: *A Conservation Law for Generalization Performance*. janeiro 1994. https://openreview.net/forum?id=HJZjx2-OWH, acesso em 2020-02-28. 5

[39] Giraud-Carrier, Christophe, Ricardo Vilalta e Pavel Brazdil: *Introduction to the Special Issue on Meta-Learning*. Machine Learning, 54(3):187–193, março 2004, ISSN 1573-0565. https://doi.org/10.1023/B:MACH.0000015878.60765.42, acesso em 2019-11-23. 6, 8

[40] Katz, Gilad, Eui Chul Richard Shin e Dawn Xiaodong Song: *ExploreKit: Automatic Feature Generation and Selection*. 2016 IEEE 16th International Conference on Data Mining (ICDM), 2016. 6

[41] Horn, Franziska, Robert Pack e Michael Rieger: *The autofeat Python Library for Automated Feature Engineering and Selection*. Em Cellier, Peggy e Kurt Driessens (editores): *Machine Learning and Knowledge Discovery in Databases*, Communications in Computer and Information Science, páginas 111–120, Cham, 2020. Springer International Publishing, ISBN 978-3-030-43823-4. 6

[42] Swearingen, Thomas, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross e Kalyan Veeramachaneni: *ATM: A distributed, collaborative, scalable system for automated machine learning*. Em *2017 IEEE International Conference on Big Data (Big Data)*, páginas 151–162, dezembro 2017. 6

[43] Kotthoff, Lars, Chris Thornton, Holger H. Hoos, Frank Hutter e Kevin Leyton-Brown: *Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA*. Em Hutter, Frank, Lars Kotthoff e Joaquin Vanschoren (editores): *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, páginas 81–95. Springer International Publishing, Cham, 2019, ISBN 978-3-030-05318-5. https://doi.org/10.1007/978-3-030-05318-5_4, acesso em 2020-08-10. 6

[44] Maher, Mohamed e Sherif Sakr: *SmartML: A Meta Learning-Based Framework for Automated Selection and Hyperparameter Tuning for Machine Learning Algorithms*, 2019. https://openproceedings.org/2019/conf/edbt/EDBT19_paper_235.pdf, acesso em 2020-08-31, Version Number: 1 type: dataset. 7

[45] Vanschoren, Joaquin: *Understanding Machine Learning Performance with Experiment Databases (Het verwerven van inzichten in leerperformantie met experiment databanken)*. maio 2010. https://lirias.kuleuven.be/1652449, acesso em 2020-02-14. 7

[46] Aha, David W.: *Generalizing from Case Studies: A Case Study*. Em *In Proceedings of the Ninth International Conference on Machine Learning*, páginas 1–10. Morgan Kaufmann, 1992. 7

[47] Rice, John R.: *The Algorithm Selection Problem*. Em Rubinoff, Morris e Marshall C. Yovits (editores): *Advances in Computers*, volume 15, páginas 65–118. Elsevier, janeiro 1976. http://www.sciencedirect.com/science/article/pii/S0065245808605203, acesso em 2020-02-18. 8

[48] Maudsley, Donald B.: *A Theory of Meta-Learning and Principles of Facilitation: An Organismic Perspective.* Tese de Doutoramento, University of Toronto, 1979. 8

[49] Lemke, Christiane, Marcin Budka e Bogdan Gabrys: *Metalearning: a survey of trends and technologies.* Artificial Intelligence Review, 44(1):117–130, junho 2015, ISSN 1573-7462. https://doi.org/10.1007/s10462-013-9406-y, acesso em 2020-01-10. 8

[50] Smith-Miles, K. A.: *Cross-disciplinary perspectives on meta-learning for algorithm selection.* ACM Computing Surveys (CSUR), janeiro 2009. https://dl.acm.org/doi/abs/10.1145/1456650.1456656, acesso em 2020-04-03, Publisher: ACM PUB27 New York, NY, USA. 8, 9

[51] Dua, Dheeru e Casey Graff: *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences, 2017. 8

[52] Vanschoren, Joaquin, Jan N. van Rijn, Bernd Bischl e Luis Torgo: *OpenML: Networked Science in Machine Learning.* SIGKDD Explorations, 15(2):49–60, 2013. 8, 25

[53] Reif, Matthias, Faisal Shafait, Markus Goldstein, Thomas Breuel e Andreas Dengel: *Automatic classifier selection for non-experts.* Pattern Analysis and Applications, 17(1):83–96, fevereiro 2014, ISSN 1433-755X. https://doi.org/10.1007/s10044-012-0280-z, acesso em 2020-04-03. 9

[54] Castiello, Ciro, Giovanna Castellano e Anna Maria Fanelli: *Meta-data: Characterization of Input Features for Meta-learning.* Em Torra, Vicenç, Yasuo Narukawa e Sadaaki Miyamoto (editores): *Modeling Decisions for Artificial Intelligence*, Lecture Notes in Computer Science, páginas 457–468, Berlin, Heidelberg, 2005. Springer, ISBN 978-3-540-31883-5. 9

[55] Morais, Gleison e Ronaldo C. Prati: *Complex Network Measures for Data Set Characterization.* Em *2013 Brazilian Conference on Intelligent Systems*, páginas 12–18, outubro 2013. 9

[56] Pimentel, Bruno e Andre de Carvalho: *A New Data Characterization for Selecting Clustering Algorithms Using Meta-Learning.* Information Sciences, 477, outubro 2018. 9

[57] Munson, M. Arthur: *A study on the importance of and time spent on different modeling steps.* ACM SIGKDD Explorations Newsletter, 13(2):65–71, maio 2012, ISSN 1931-0145. https://doi.org/10.1145/2207243.2207253, acesso em 2020-08-29. 20

[58] García, Salvador, Julián Luengo e Francisco Herrera: *Tutorial on practical tips of the most influential data preprocessing algorithms in data mining.* Knowledge-Based Systems, 98:1–29, abril 2016, ISSN 09507051. https://linkinghub.elsevier.com/retrieve/pii/S0950705115004785, acesso em 2020-09-01. 20

[59] Zhu, Xingquan e Xindong Wu: *Class Noise vs. Attribute Noise: A Quantitative Study.* Artificial Intelligence Review, 22(3):177–210, novembro 2004, ISSN 1573-7462. https://doi.org/10.1007/s10462-004-0751-8, acesso em 2020-09-04. 21

[60] Frenay, B. e M. Verleysen: *Classification in the Presence of Label Noise: a Survey*. Neural Networks and Learning Systems, IEEE Transactions on, 25(5):845 – 869, 2014. 21

[61] Wilson, Dennis L.: *Asymptotic Properties of Nearest Neighbor Rules Using Edited Data*. IEEE Transactions on Systems, Man, and Cybernetics, SMC-2(3):408–421, julho 1972, ISSN 2168-2909. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics. 21

[62] Sluban, Borut, Dragan Gamberger e Nada Lavra: *Advances in Class Noise Detection*. Em *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, páginas 1105–1106, agosto 2010, ISBN 978-1-60750-605-8. 21

[63] Longadge, Rushi e Snehalata Dongre: *Class Imbalance Problem in Data Mining Review*. arXiv:1305.1707 [cs], maio 2013. `http://arxiv.org/abs/1305.1707`, acesso em 2020-04-07, arXiv: 1305.1707. 21

[64] Chawla, N. V., K. W. Bowyer, L. O. Hall e W. P. Kegelmeyer: *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research, 16:321–357, junho 2002, ISSN 1076-9757. `http://arxiv.org/abs/1106.1813`, acesso em 2020-07-10, arXiv: 1106.1813. 21

[65] Zhang, J. e I. Mani: *KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction*. Em *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*, 2003. 21

[66] Seliya, N., T. M. Khoshgoftaar e J. Van Hulse: *A Study on the Relationships of Classifier Performance Metrics*. Em *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, páginas 59–66, novembro 2009. ISSN: 2375-0197. 21, 22

[67] Joshi, M.V.: *On evaluating performance of classifiers for rare classes*. Em *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, páginas 641–644, dezembro 2002. ISSN: null. 22

[68] Bousquet, Olivier e André Elisseeff: *Stability and Generalization*. Journal of Machine Learning Research, 2(Mar):499–526, 2002, ISSN ISSN 1533-7928. `http://www.jmlr.org/papers/v2/bousquet02a.html`, acesso em 2020-02-18. 22

[69] Xu, Huan, Constantine Caramanis e Shie Mannor: *Sparse Algorithms Are Not Stable: A No-Free-Lunch Theorem*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(1):187–193, janeiro 2012, ISSN 1939-3539. 22

[70] Japkowicz, Nathalie: *Classifier evaluation: A need for better education and restructuring*. janeiro 2008. 22

[71] Sokolova, Marina, Nathalie Japkowicz e Stan Szpakowicz: *Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation*. Em Sattar, Abdul e Byeong ho Kang (editores): *AI 2006: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, páginas 1015–1021, Berlin, Heidelberg, 2006. Springer, ISBN 978-3-540-49788-2. 22

[72] Elrahman, Shaza M Abd e Ajith Abraham: *A Review of Class Imbalance Problem.* Journal of Network and Innovative Computing, 1:332–340, 2013, ISSN 2160-2174. 23

[73] Brodersen, K. H., C. S. Ong, K. E. Stephan e J. M. Buhmann: *The Balanced Accuracy and Its Posterior Distribution.* Em *2010 20th International Conference on Pattern Recognition*, páginas 3121–3124, agosto 2010. ISSN: 1051-4651. 23

[74] Mosley, Lawrence: *A balanced approach to the multi-class imbalance problem.* Graduate Theses and Dissertations, janeiro 2013. `https://lib.dr.iastate.edu/etd/13537`. 23

[75] Friedman, Jerome H.: *Greedy Function Approximation: A Gradient Boosting Machine.* The Annals of Statistics, 29(5):1189–1232, 2001, ISSN 0090-5364. `https://www.jstor.org/stable/2699986`, acesso em 2020-07-10, Publisher: Institute of Mathematical Statistics. 26

[76] Ho, Tin Kam: *The random subspace method for constructing decision forests.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):832–844, agosto 1998, ISSN 1939-3539. 26

[77] Breiman, Leo: *Random Forests.* Machine Learning, 45(1):5–32, outubro 2001, ISSN 1573-0565. `https://doi.org/10.1023/A:1010933404324`, acesso em 2020-07-10. 26

[78] Chan, T. F., G. H. Golub e R. J. LeVeque: *Updating Formulae and a Pairwise Algorithm for Computing Sample Variances.* Em *COMPSTAT*, páginas 30–41, 1982, ISBN 978-3-642-51461-6. 26

[79] Quinlan, J. R.: *Induction of decision trees.* Machine Learning, 1(1):81–106, março 1986, ISSN 1573-0565. `https://doi.org/10.1007/BF00116251`, acesso em 2020-07-10. 26

[80] Friedman, Jerome, Trevor Hastie e Robert Tibshirani: *The elements of statistical learning*, volume 1. Springer Series in Statistics, New York, 2001. 26

[81] Fletcher, R.: *Practical methods of optimization.* Wiley, Chichester ; New York, 2nd ed edição, 1987, ISBN 978-0-471-91547-8. 26

[82] Cristianini, Nello e John Shawe-Taylor: *An introduction to support vector machines: And other kernel-based learning methods.* Cambridge University Press, Cambridge; New York, 2013, ISBN 978-1-139-64908-7 978-0-511-80138-9 978-1-139-63862-3. OCLC: 1031088649. 26

[83] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot e Édouard Duchesnay: *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12(85):2825–2830, 2011. `http://jmlr.org/papers/v12/pedregosa11a.html`, acesso em 2020-07-10. 26

[84] Demšar, Janez: *Statistical Comparisons of Classifiers over Multiple Data Sets.* Journal of Machine Learning Research, 7:1–30, dezembro 2006. 34

Anexo I

Artigo Submetido ao AAAI 2021 (Qualis A1)

Pipeline Recommendation Based on Meta-Learning

Juliana Hosoume, [1] **Victor H. Barella,** [2] **André C. P. L. F. de Carvalho,** [2] **Luís P. F. Garcia** [1]
[1]Computer Science Department, University of Brasília, Brasília, Distrito Federal, Brazil
[2]Institute of Mathematics and Computer Science, University of São Paulo, São Carlos, São Paulo, Brazil
jhosoume@gmail.com
{victorhb, andre}@icmc.usp.br
luis.garcia@unb.br

Abstract

Machine learning has a multitude of algorithms and preprocessing techniques that address classification problems. Combining them to design the best data classification pipeline is a challenging task. Different approaches have been investigated, including handmade pipelines, Bayesian optimization and genetic programming. Nevertheless, each one of these approaches has hindrances, as the need for a human specialist for handmade pipelines, or the computational cost of Bayesian optimization, and genetic programming. Meta-learning can overcome these drawbacks by using the pipeline knowledge accumulated from previous experiments. Thus, the knowledge is stored in a meta-base whose predictive attributes are meta-features extracted from datasets, and the target attributes represent the predictive performance of successful pipelines applied to these datasets. This study proposes the use of meta-learning as a pipeline builder to predict the performance of combinations of preprocessing techniques, like noise detection and unbalanced algorithms for classification problems. For such, a meta-base composed of 130 meta-features and over 390 datasets were used to induce meta-regressors with different biases. The meta-regressors predicted the balanced accuracy of the combinations with low error, and some outperformed the baselines. After selecting the most suitable meta-regressor, the recommender system was enhanced to recommend a pipeline composed of multiple preprocessing steps. According to the experimental results, the proposed strategy performed better than the baselines.

Introduction

Over the years, the number of available datasets has increased, concomitantly to computational power. In this scenario, complex data analyses were facilitated. Nonetheless, defining the best methods to apply on a particular dataset is still an exhaustive task, and it usually involves using Machine Learning (ML) pipelines. The definition of a proper ML pipeline requires not only choosing a suitable ML algorithm, but also selecting preprocessing techniques. Furthermore, each of these steps is data sensitive, i.e., the performance depends on the characteristics of the dataset (Wolpert 1992; Adam et al. 2019).

Finding a suitable pipeline for a novel ML problem depends on expertise and is very time demanding. An Automated ML (AutoML) system could aid non-experts on this task (Guyon et al. 2016). Different approaches enables this search, such as genetic programming (Olson and Moore 2019) and Bayesian optimization (Thornton et al. 2013). However, each of them has backwardness from human interference to computational cost. Meta-learning (MtL) can overcome these drawbacks with the concept of meta-knowledge (Laadan et al. 2019).

MtL addresses the problem of technique selection by making use of accumulated knowledge of prior tasks, or meta-knowledge (Brazdil et al. 2009). This approach has been employed in recommender systems (Vanschoren et al. 2012), Data Mining (DM) systems with intelligent agents (Prodromidis et al. 2000) and hyperparameter optimization (Feurer, Springenberg, and Hutter 2015). Although it has obtained good results in predicting a classifier's performance based on statistical and geometrical characteristics of a dataset (Garcia et al. 2018), a smaller number of studies focused on data preprocessing (Schoenfeld et al. 2018; Garcia, Carvalho, and Lorena 2016; Bilalli et al. 2018).

MtL goes beyond traditional base-learning (Vilalta, Giraud-Carrier, and Brazdil 2010). Commonly, the first step in using MtL consists of building a meta-base composed of meta-examples (Vanschoren 2018). Each meta-example includes meta-features and a label for each ML algorithm used in a study. Meta-features are a set of characteristics extracted from a dataset. The label can be a measure of the performance of a base-model.

After building a meta-dataset with enough meta-examples, the next step is inducing a meta-model. The meta-model goal is to predict the behavior of a base-model for a new dataset, using only its meta-features. Therefore, besides recommending the algorithm with the best predictive performance for the dataset, a ranking or list of performance measures can be obtained by applying the meta-model (Brazdil et al. 2009). However, occasionally the meta-model is not sufficiently accurate. Previous works suggest the need of a preprocessing step, such as data imbalance treatment (Muñoz, M. A. et al. 2018).

In this work, a pipeline recommendation design for AutoML is investigated. We focus on the noise detection and balancing classes algorithms for the preprocessing step. In this sense, a MtL approach is used to tackle the complex task of finding a proper combination of multiple preprocessing techniques and a classifier for a new ML problem. For such, state-of-the-art meta-features (Rivolli et al. 2019) are used to induce a suitable meta-regressor to predict the per-

formance of the pipelines. A repeated hit analysis on multiple steps of recommendation is used to evaluate the building of the pipeline Finally, the results are compared to baselines to assess the efficiency of the proposed methodology. In the experimental results, a Random Forest based recommendation performed better than the default baseline, in both the base analysis and in the pipeline construction.

Related Works

Automated Machine Learning

Defining an optimal sequence of methods for data mining is not a simple task. There are plenty of different preprocessing and ML algorithms that can be combined to implement a ML solution with a good predictive performance for a novel dataset. Nonetheless, as stated in the No Free Lunch theorems (Wolpert and Macready 1997; Adam et al. 2019), no algorithm can perform well on all ML problems, hence much time and effort are demanded. The Automated Machine Learning (AutoML) process aims at easing the task of selecting suitable pipelines and improve productivity.

An AutoML system can focus on different ML problems. Some studies target at feature selection and generation (Katz, Shin, and Song 2016; Horn, Pack, and Rieger 2020) as a key step for the success of a ML solution. Others explore the model selection and hyperparameter optimization step (Swearingen et al. 2017; Kotthoff et al. 2019). The recommendation of a complete workflow is also investigated in recent studies (Olson and Moore 2019; Laadan et al. 2019). The definition of a entire pipeline is a daunting task. The search space of possible pipelines is extensive and each component choice requires its own expertise. Besides, the algorithms performance depends on the input and outcome of each pipeline component influences on the next steps, thus increasing the task complexity (Laadan et al. 2019).

Many different approaches of AutoML have been proposed. Among them is genetic programming, implemented in the TPOT (Olson and Moore 2019), which is a method based on generation and evolution of pipelines. Alternatively, Auto-sklearn (Feurer et al. 2015) and auto-WEKA (Thornton et al. 2013) employ Bayesian Optimization, which is centered on probabilistic models to predict algorithm performance. One other option to tackle the problem of searching a pipeline is to use MtL, as applied in the RankML (Laadan et al. 2019).

In this work, the proposed method extends previous studies on MtL approach on AutoML (Maher and Sakr 2020) by considering recommendation of multiple preprocessing steps pipeline. Besides, it differs from the RankML system (Laadan et al. 2019) and AutoML (Feurer et al. 2015) in the process of evaluation and building the suitable pipelines. Whilst RankML uses pipeline representation and AutoML applies MtL combined with Bayesian optimization, here we build the pipeline in an iterative manner based only on meta-characterization of the datasets, without storing information of the pipeline structure. Therefore, reducing possible overhead and simplifying the representation of the investigated problem.

Meta-Learning

Firstly envisioned by Aha (1992), MtL is centered on learning the outcome by knowledge of known processes. For such, MtL focus on the output of algorithms and procedures, not on bias or internal differentiation of the learning algorithms (Prodromidis et al. 2000). MtL fits in the Rice's algorithm selection model (Rice 1976). In this abstraction, there are four main components: the problem space P, the feature space F, the algorithm space A and the performance space Y. They represent the set of possible problem instances, a collection of measurable characterization of the instances, a chosen set of suitable algorithms that could solve the problem and performance metrics for each combination of algorithms and instances, respectively. A system that employs MtL is based on a procedure to map a problem $p \in P$, characterized by meta-features $f \in F$, to an algorithm $\alpha \in A$, which is the most suitable algorithm for the task according to the metric $y \in Y$, $y(\alpha(p))$ (Smith-Miles 2009).

In this context, the complexity of these problems can be measured by the extension and dimensionality of each set and the intricacy of the mappings. For instance, the problem space P may contain different tasks from multiple domains. Ideally, this set should be composed of as many diverse problems as the MtL system would be presented. These problem datasets can be gathered from the literature or open-data repositories as UCI (Dua and Graff 2017) or OpenML (Vanschoren et al. 2013).

Meta-Features The selected set of meta-features (F) is critical to the success of the meta-model. This set should convey enough information about the investigated tasks. Besides, an indispensable attribute of a ML solution is the reproducibility. To this end, the selection of meaningful predictive attributes that enables replication of all calculation steps is fundamental to a recommendation system based on MtL (Alcobaça et al. 2020; Rivolli et al. 2019).

Following the formal definition presented by Alcobaça et al. (2020) and Rivolli et al. (2019), meta-features are a set of k values extracted from a dataset D. Therefore, this process can be modeled as a function $f : D \rightarrow \mathbb{R}^k$, $f(D) = \theta(m(D, h_m), h_\theta)$. This equation highlights two main components of meta-features, the characterization measures, m and the summarization functions, θ. Both are dependent on the hyperparameters h_m and h_θ, respectively. The method of calculation of some groups of measures are defined by these hyperparameters, others, such as the simple meta-features, are calculated directly.

The meta-features can be classified into five groups: simple, information-theoretic, model-based, statistical and ladmarking(Reif et al. 2014; Rivolli et al. 2019).

- **Simple** meta-features, also known as general meta-features, are the most commonly used. They are easily and directly calculated from the datasets, thus require low computational resources (Reif et al. 2014). Some of them are the number of attributes, the number of binary attributes and number of categorical attributes.

- **Information-Theoretic** meta-features are measures based on entropy calculation and information-theory (Reif et al. 2014).

- **Model-Based** meta-features are obtained from a ML model trained on the data. Usually, these measures are extracted from properties of Decision Tree (DT) models (Reif et al. 2014).

- **Statistical** meta-features assess statistical indicators of the data. We consider here only numerical information. Average, correlation, standard deviation are some examples of these statistical measure (Reif et al. 2014; Castiello, Castellano, and Fanelli 2005) .

- **Landmarking** meta-features based on the predictive performance of a set of ML algorithms. In order to appropriately define a dataset, these algorithms should have diverse biases and low computational cost (Smith-Miles 2009).

In some categorizations, there is a sixth group composed of non-traditional meta-features measures. Usually, these meta-features are complex, computationally costly or domain specific (Rivolli et al. 2019). Nonetheless, for some situations, they have shown promising results (Garcia et al. 2018; Morais and Prati 2013; Pimentel and de Carvalho 2018).

Base-Learners and Performance Evaluation The algorithm space A is the set of candidate algorithms in the recommendation process. The performance of each algorithm is evaluated according to a selected measure, therefore the performance evaluation measures should be suitable to the problem domain. Although the algorithms are typically either only classifiers or only regressors, they should have a diversity of biases. In this context, the base-learner can be defined as a pipeline composed of preprocessing techniques and an estimator. The preprocessing step of a ML pipeline is necessary and usually very expensive in terms of time. Typically, preprocessing of data takes more than 40% of the time spent on a ML task (Munson 2012).

Although researchers can investigate a diverse range of problems nowadays due to computational power and storage, they still need to transform raw data into structured, mathematically feasible, and computationally suitable formats (Bilalli et al. 2018). The challenge increases, considering that different data sources are heterogeneous and have their peculiarities.

Many algorithms have been developed to tackle problems on the preprocessing level. Instance generation, discretization, imbalance data treatment, feature selection and noise filtering are examples of approaches commonly applied during preprocessing (García, Luengo, and Herrera 2016). Selecting more than a pair of approaches of each category to build a pipeline would increase exponentially the search space and the complexity of the algorithm space. Therefore, in this work, we focus on noise filters and imbalance data treatment.

In some cases, erroneous labels may be assigned to instances, thus resulting in class or label noise. Other times, the quality of the collected data may be poor, e.g. sensor noise, which generates attribute noise (Zhu and Wu 2004; Frenay and Verleysen 2014). In this work, we only consider the label noise problem, which is the most studied type of noise.

Label noise filters are algorithms to detect and remove label noise using different approaches as density information, descriptors of data or ensembles of classifiers. The Edited Nearest Neighbours(ENN) (Wilson 1972) algorithm is an example of a similarity or density information based filter. Another commonly used method, High Agreement Random Forest filter (HARF) (Sluban, Gamberger, and Lavra 2010), uses the Random Forest classifier to identify noise instances.

Data imbalance is a fairly common problem. It occurs when one or more classes are much less frequent in the dataset than other classes in a classification task (Longadge and Dongre 2013). To better represent the minority classes, a frequent approach is to resample the dataset. This can be done in two ways: either by oversampling the minority class, e.g. the Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al. 2002), or by undersampling the majority class, e.g. the random undersampling (RU) strategy (Zhang and Mani 2003).

Even though accuracy is a widely used measure of classifier performance, it is not suitable to evaluate imbalanced datasets. The predictive accuracy measure is biased towards the majority class (Elrahman and Abraham 2013). Hence, the balanced accuracy measure can be used to improve the evaluation in such cases (Brodersen et al. 2010).

Methodology

When MtL is used, the quality of the meta-features chosen is key in the prediction of a pipeline performance. Besides, the cost of measuring the meta-features should not overextend the demanded training and scoring time of the classifiers. In the context of AutoML, the goal is to develop a robust recommendation system of pipelines for different classification problems, based on reproducible and feasible characterization calculations.

In this work, for each chosen dataset, three main steps were performed. First, meta-features were extracted. Afterward, the dataset was preprocessed. In the last step, the averaged cross-validated balanced accuracy of the selected combinations were calculated. The combination of the meta-features and the predictive performance form the meta-base. Next, we trained the meta-learners and evaluated them on the pipeline recommendation task (Figure 1). We also made available the implementation of the recommender system[1].

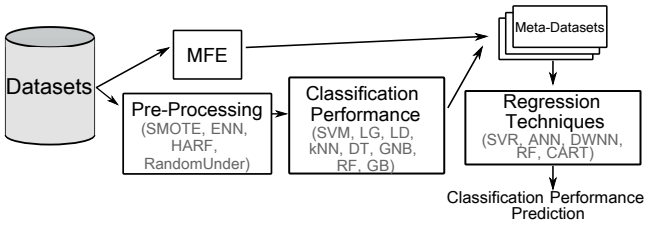


Figure 1: Steps for the construction of the meta-base and evaluation of the meta-regressors on the MtL setup.

A total of 130 different meta-features were selected to characterize 391 diverse classification datasets. The pymfe package (Alcobaça et al. 2020) was used to extract the meta-features. We used default parameters for all meta-features on pymfe, and they were summarized by mean and standard deviation. The 391 datasets were retrieved from the OpenML repository [2] (Vanschoren et al. 2013), considering a broad

[1]

[2]https://www.openml.org/

range of different areasEach dataset has a maximum of 10 classes, 10000 samples and 500 attributes. All datasets have no missing values.

Four preprocessing techniques were chosen: two noise detection filters, ENN and HARF, and two re-sampling techniques for imbalanced data, RU and SMOTE. We chose techniques with different biases to increase diversity. We also considered the datasets without any preprocessing as a possible recommendation.

In addition to the preprocessing techniques, the proposed system also recommends a classification algorithm. The set of classification algorithms in our implementation are Gradient Boosting (GB) (Friedman 2001), Random Forest (RF) (Ho 1998; Breiman 2001), Gaussian Naive Bayes (GNB) (Chan, Golub, and LeVeque 1982), Decision Tree (DT) (Quinlan 1986), k-Nearest Neighbours (kNN) (Mitchell 1997), Linear Discriminant (LD) (Friedman, Hastie, and Tibshirani 2001), Logistic Regression (LR) (Fletcher 1987), Support Vector Machines (SVM) (Cristianini and Shawe-Taylor 2013). All classifiers used the default parameters presented in the Scikit-learn module (Pedregosa et al. 2011).

On the MtL level, we trained regressors to estimate the predictive performance of each possible combination of a pre-processing technique and a classification algorithm. For a given dataset, the combination with the highest estimated balanced accuracy is recommended by the system. We considered five regression algorithms with different biases: Artificial Neural Network (ANN), Decision Tree (CART), Distance Weighted k-Nearest Neighbout (DWNN), Random Forest Regressor (RFR) and Support Vector Regressors (SVR).

To evaluate the performance of the regressors, we considered two different analysis, the meta-level analysis and the base-level analysis. On the meta-level analysis, we measure the error of the regressors on estimating the predictive performance of each possible combination by means of RMSE. On the base-level analysis, we assess the overall gain of using the recommended combination compared to two baselines: a random choice of combination (RD) and the combination with the highest balanced accuracy in the training set (DF). Both the meta-level and base-level analysis were evaluated using 10-fold cross-validation.

We also evaluated the performance of our system on recommending a pipeline, consisting of none, one or more preprocessing techniques and a classification algorithm. Due to computational cost, we limited the number of preprocessing techniques to be sequentially applied to a dataset to a maximum of 5 iterations. The datasets were split into two groups: 90% for training and 10% for testing. The regressor with the best performance on the base analysis was trained on the first aforementioned group. To properly asses the best pipeline for each testing dataset, we directly compute the performance of a simple combination of one preprocessing and a classifier in an iterative manner.

Afterward, for each testing dataset, we used our system to recommend, at each step of the pipeline, a combination. If the recommended combination contains a preprocessing technique, the technique is added to the recommended pipeline. Thus, the preprocessing technique is applied to the dataset and the meta-features are extracted anew. If not, the classification algorithm is added to the pipeline and the system returns the pipeline as a recommendation. The process continues until one of the three conditions indicates the halting point. These conditions are: no preprocessing technique is recommmended, the number of preprocessing techniques reaches 5 iterations, or the recommended technique is different from the best pipeline previously measured in the iteration. The same procedure was applied to a default baseline, which, for all datasets and all rounds, indicates the combination with the maximum mean performance in the meta-base. This experiment was performed 10 independent times for each meta-regressor.

All calculations were run in a cluster node with two Intel Xeon Silver 4114 processors of 2.20GHz with 256 Gb of RAM. The installed operating system is Debian OS version 9. The programming languages used to implement the experiments were Python 3 and R. The main package used was Scikit-learn.

Results

In this section, we present the experimental results for the four analysis, meta-base, meta-level, base-level, and pipeline evaluation. On the meta-base analysis, we show the performance of each pair of combinations of one preprocessing technique and one classification algorithm, on average and number of wins compared to the others. On the meta-level analysis, the performance measures of the regressors in the task of predicting the balanced accuracy of each possible combination of pairs are described. On the base-level analysis, we compare our recommender system with RD and DF baselines. We also present an analysis of the meta-features importance regarding the RF regressors. On the pipeline analysis, we analyze the number of correct predicted technique at each round of the pipeline according to a previously measured combination performance.

Meta-Base Analysis

In Figure 2, the mean balanced accuracy for each combination of preprocessor and classifier is presented. The darker the color, the better that combination performed on the datasets on average. The combination of SMOTE and RU with GB attained the most promising results. Meanwhile, SVM combined with any preprocessing technique obtained the worst results, especially for no preprocessing and ENN. Nonetheless, since all combinations performed above 60% of balanced accuracy in average, they were considered suitable for the following analysis.

The average performance of techniques is not enough to justify recommending a combination. We should still strive for diversification in combinations bias, which could lead to a competition between the combinations, with each one of them performing better than the others on, at least, a small subset of datasets. We show here that, although some combinations perform better in general, other combinations still perform better on a subset of datasets.

Figure 3 shows the number of times a combination obtained the highest balanced accuracy for each dataset, which we call number of wins. When more than one combination presented the highest balanced accuracy value, a win is added to both. Similar to the results in Figure 2, GB was the winner classifier, with a number of 135 wins. Regarding the

	None	SMOTE	RandomUnder	HARF	ENN
GB	0.75 ± 0.20	0.78 ± 0.19	0.78 ± 0.19	0.76 ± 0.20	0.74 ± 0.20
RF	0.75 ± 0.20	0.77 ± 0.19	0.78 ± 0.19	0.75 ± 0.20	0.72 ± 0.21
GNB	0.68 ± 0.18	0.70 ± 0.18	0.70 ± 0.18	0.69 ± 0.18	0.68 ± 0.19
DT	0.72 ± 0.19	0.75 ± 0.19	0.74 ± 0.19	0.74 ± 0.19	0.72 ± 0.20
kNN	0.66 ± 0.20	0.70 ± 0.19	0.69 ± 0.19	0.68 ± 0.20	0.67 ± 0.20
LD	0.68 ± 0.19	0.71 ± 0.18	0.71 ± 0.19	0.68 ± 0.19	0.67 ± 0.19
LR	0.67 ± 0.20	0.71 ± 0.19	0.71 ± 0.19	0.68 ± 0.21	0.67 ± 0.20
SVM	0.60 ± 0.21	0.64 ± 0.22	0.64 ± 0.21	0.61 ± 0.21	0.60 ± 0.21

Figure 2: Meta-base analysis. Mean and standard deviation of balanced accuracies for a combination of a preprocessor and a classification algorithm.

preprocessing techniques, SMOTE won the most, with a total of 202 wins. In this scenario, every combination of classifier and preprocessing has at least five wins.Nonetheless, the meta-base is class imbalanced, since combinations that involves GB, RF and SMOTE concentrate most of the winning examples.

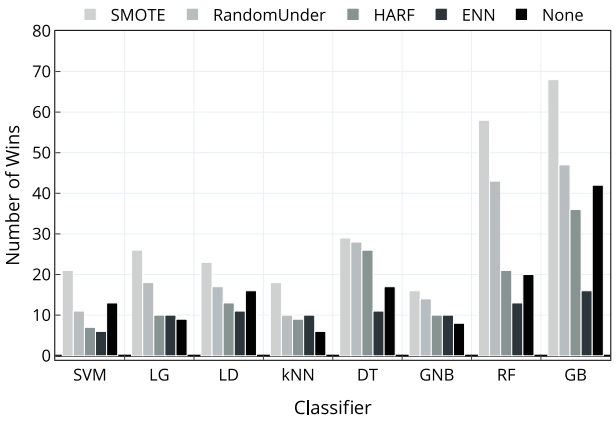


Figure 3: Distribution of wins for each combination of a preprocessing technique and a classification algorithm.

Meta-Level Analysis

We approached the recommendation task as a regression task. In this regard, we build regressors to predict the performance of each combination giving a set of meta-features. Therefore, a total of 200 meta-regressors are needed for each iteration in the cross-validation methodology. Figure 4 shows the average root mean squared error (RMSE). Each plot groups values obtained for one preprocessor and each boxplot summarizes the estimated results of a meta-regressor (in white) over all base-classifiers. The RMSE was calculated by a ten-fold cross-validation process. The last two boxplots (in gray) of each plot, RD and DF, are the baselines against which the meta-regressors are compared to.

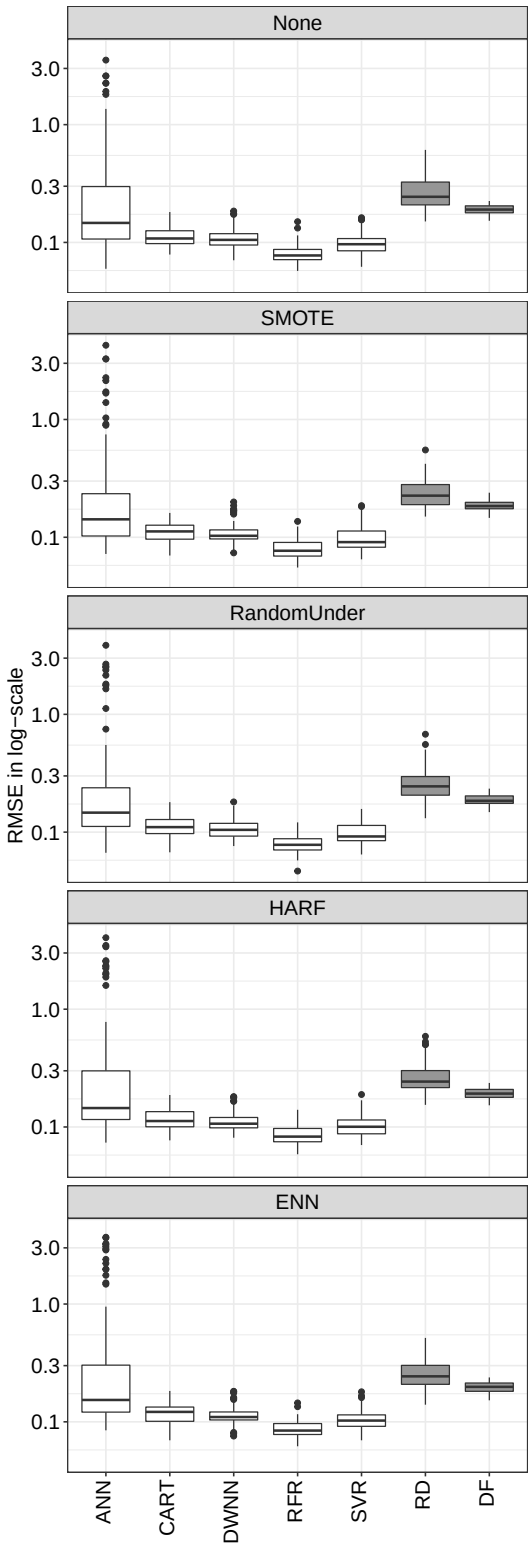


Figure 4: RMSE of each meta-regressor for each preprocessor, grouping the base-classifiers results.

Using the Friedman statistical test with the Nemenyi post-test at 95% confidence level (Demšar 2006), the CART, DWNN and RFR presented better predictive performance than the baselines, which indicates that they predicted the performance of the pipelines more accurately. Nonetheless, ANN was as error prone as both the baselines. RFR performed better than the other meta-regressors. There is no significant difference in performance considering the preprocessors.

Base-Level Analysis

Next, we evaluate the recommendations given by the MtL-based system. For each regression algorithm, the system recommends the combination with the highest predictive performance. Figure 5 shows the gain or loss in percentage of the recommendations over each baseline. The difference between the performances of the recommendations given by each regressor and the ones given by the baselines baselines for all datasets are summed and, then, normalized by the sum of gains of the best possible pipeline. This prediction was performed ten times. Whilst ANN and CART only outperform the random baseline, DWNN, RFR and SVR have better gains in balanced accuracy than both RD and DF. In this sense, collecting the meta-features from a dataset, followed by the subsequent use of the recommendation system, leads to an increased balanced accuracy in the classification task by applying the recommended combination. This result augments the previous ones (Garcia et al. 2018), since the recommendation not only considers classification algorithms, but also the intricate process of choosing a proper preprocessing technique.

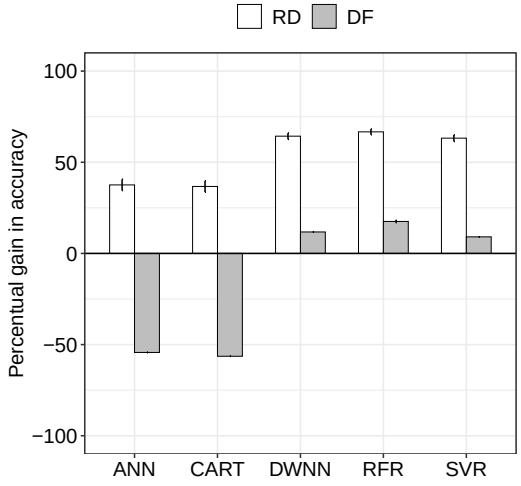


Figure 5: Improvement of base-classifier base accuracy over baselines when normalized by the best balanced accuracies in the meta-base.

Random Forest Feature Importance

In order to better understand which meta-features contributed more to achieve the results, Figure 6 shows the 20th-est most important meta-features to RFR, according to their individual contribution to the reduction of the mean squared error (MSE), as well as their corresponding meta-feature groups. The contribution was measured as the increase in MSE when that meta-feature was omitted. the x-axis lists the meta-features in decreasing order of relevance, with the mean change in MSE shown in the y-axis. Vertical bars indicate standard errors of estimation.

The meta-feature with the highest impact was the model-based nodes per instance characterization, a measure of the ratio of non-leaf nodes per number of instances in a decision tree model. The landmarking group of meta-features is amid the most relevant, followed by the clustering measures.

Pipeline Construction

The effectiveness of the recommendation system to predict a suitable pipeline composed of multiples steps of preprocessing is shown in Figure 7. At least 40% of correct suggestions were obtained for the testing datasets in the first round, considering preprocessing techniques and classification algorithms predictions independently. Almost 25% of the pipelines predicted in the first round were fully correspondent to the pipelines with the best predictive performance. When compared to the baseline, the recommendation system based on RF was better in all cases, except for the recommendation of preprocessing techniques in the first round.

For both, RF and DF, the number of datasets evaluated decreased at each iteration, since the prediction of preprocessing methods did not match with the best measured pipeline, or the model reported that no further preprocessing was needed. In light of this result, one hypothesis is that not only the consecutive application of preprocessing techniques as noise filters could degenerate the datasets further in the pipeline but also the datasets that already have its pipelines correctly predicted in earlier stages could be less complex, thus leading to increased hit rate in late rounds.

Conclusions

In this paper, a MtL-based system to recommend a combination of preprocessing technique with a classification algorithm was proposed and experimentally assessed. We also investigated the recommendation of a pipeline composed of multiple preprocessing steps. An extensive and reproducible meta-features extraction process was performed to characterize a heterogeneous set of classification tasks from different domains. In this context, meta-regressors were trained to predict the expected balanced accuracy of combinations of four preprocessing techniques and eight different classification algorithms.

The meta-regressors performances in this problem were compared to two baselines, default and random. RFR was the best meta-model in all cases, and, therefore, selected as regression algorithm for the pipeline recommendation system. In the pipeline construction task, models induced by RFR were able to outperform a default strategy, thus recommending properly a complex pipeline for approximately 40% of the datasets.

As future work, we expect to further analyze other regressors bias and reduce the number of necessary meta-features. Besides, augmenting the system to a ranking approach would benefit an informed decision on the most suit-

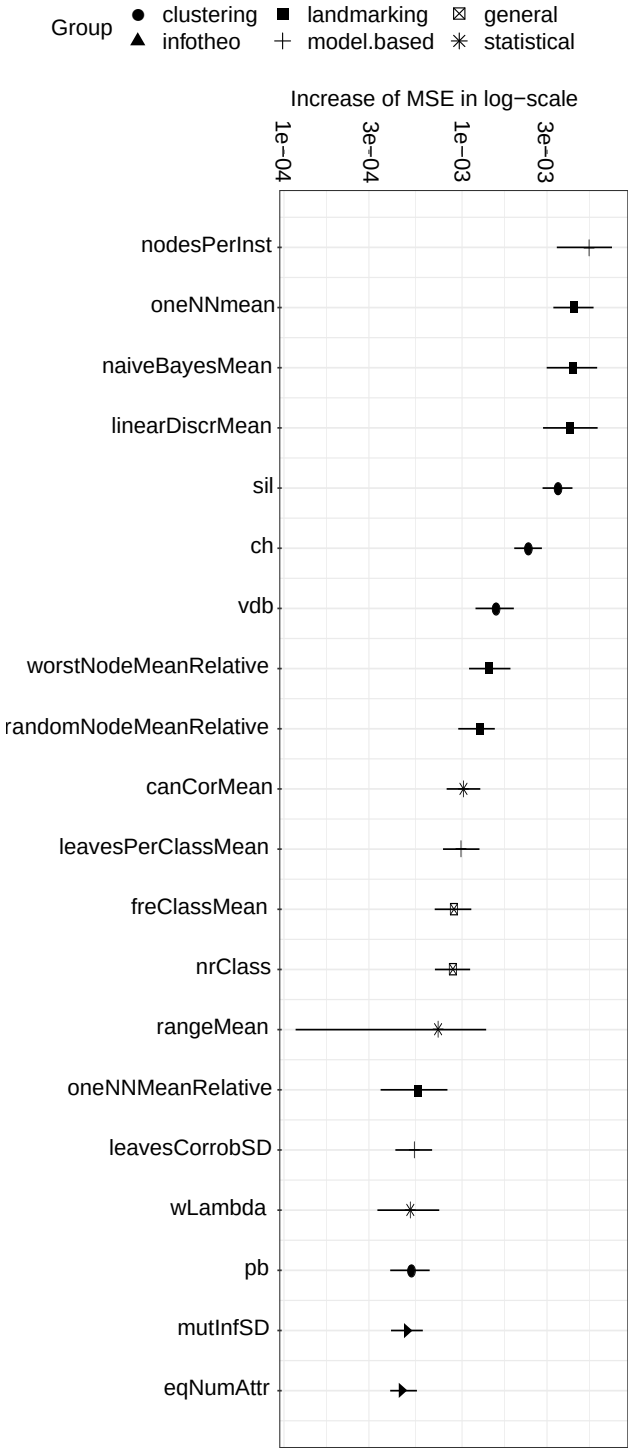
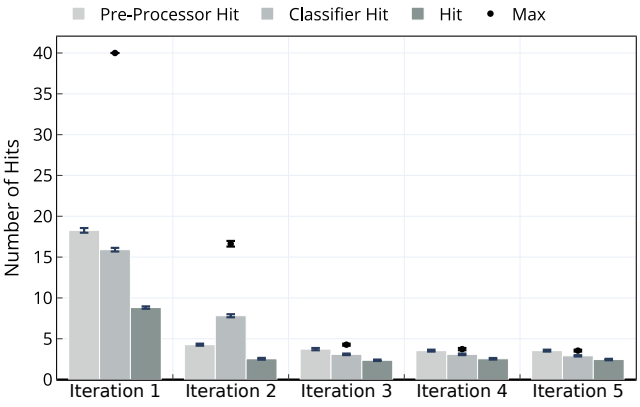
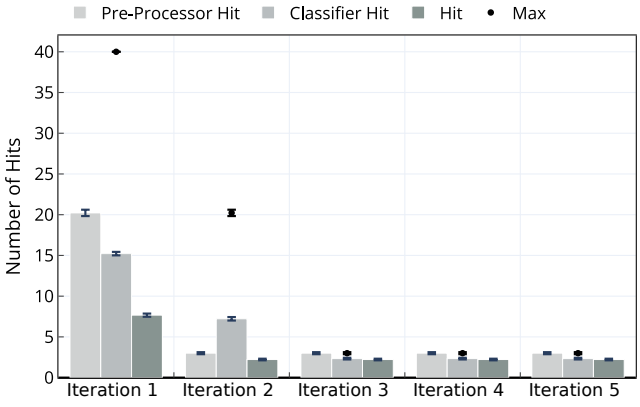


Figure 6: Feature impact on the prediction capabilities of the Random Forest regressor.



(a) Random Forest



(b) Default

Figure 7: Meta-regressor number of hits for each round. The bars indicates the number of correctly predicted pre-processor, classifier and combination of the two (hit). The dot (max) indicates the number of datasets evaluated in each round.

able pipeline for a given classification task. Specifically, we plan to investigate the impact of repeatedly preprocessing a dataset.

Acknowledgments

References

Adam, S. P.; Alexandropoulos, S.-A. N.; Pardalos, P. M.; and Vrahatis, M. N. 2019. No Free Lunch Theorem: A Review. Springer Optimization and Its Applications, 57–82. Springer International Publishing.

Aha, D. W. 1992. Generalizing from Case Studies: A Case Study. In *In Proceedings of the Ninth International Conference on Machine Learning*, 1–10.

Alcobaça, E.; Siqueira, F.; Rivolli, A.; Garcia, L. P. F.; Oliva, J. T.; and Carvalho, A. C. P. L. F. d. 2020. MFE: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research* 21(111): 1–5.

Bilalli, B.; Abelló, A.; Aluja-Banet, T.; and Wrembel, R. 2018. Intelligent assistance for data pre-processing. *Computer Standards & Interfaces* 57: 101–109.

Brazdil, P.; Carrier, C. G.; Soares, C.; and Vilalta, R., Brazdil, P.; Carrier, C. G.; Soares, C.; and Vilalta, R. 2009. *Metalearning: Applications to Data Mining*. Cognitive Technologies.

Breiman, L. 2001. Random Forests. *Machine Learning* 45(1): 5–32.

Brodersen, K. H.; Ong, C. S.; Stephan, K. E.; and Buhmann, J. M. 2010. The Balanced Accuracy and Its Posterior Distribution. In *2010 20th International Conference on Pattern Recognition*, 3121–3124.

Castiello, C.; Castellano, G.; and Fanelli, A. M. 2005. Meta-data: Characterization of Input Features for Meta-learning. In Torra, V.; Narukawa, Y.; and Miyamoto, S., eds., *Modeling Decisions for Artificial Intelligence*, Lecture Notes in Computer Science, 457–468. Berlin, Heidelberg: Springer.

Chan, T. F.; Golub, G. H.; and LeVeque, R. J. 1982. Updating Formulae and a Pairwise Algorithm for Computing Sample Variances. In *COMPSTAT*, 30–41.

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; and Kegelmeyer, W. P. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16: 321–357.

Cristianini, N.; and Shawe-Taylor, J., Cristianini, N.; and Shawe-Taylor, J. 2013. *An introduction to support vector machines: And other kernel-based learning methods*. Cambridge; New York: Cambridge University Press.

Demšar, J. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* 7: 1–30.

Dua, D.; and Graff, C., Dua, D.; and Graff, C. 2017. *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences.

Elrahman, S. M. A.; and Abraham, A. 2013. A Review of Class Imbalance Problem. *Journal of Network and Innovative Computing* 1: 332–340.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, 2962–2970.

Feurer, M.; Springenberg, J. T.; and Hutter, F. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Fletcher, R., Fletcher, R. 1987. *Practical methods of optimization*. Chichester ; New York: Wiley, 2nd ed edition.

Frenay, B.; and Verleysen, M. 2014. Classification in the Presence of Label Noise: a Survey. *Neural Networks and Learning Systems, IEEE Transactions on* 25(5): 845 – 869.

Friedman, J.; Hastie, T.; and Tibshirani, R., Friedman, J.; Hastie, T.; and Tibshirani, R. 2001. *The elements of statistical learning*, volume 1. Springer Series in Statistics, New York.

Friedman, J. H. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* 29(5): 1189–1232.

Garcia, L. P. F.; Carvalho, A. C. P. L. F. d.; and Lorena, A. C. 2016. Noise detection in the meta-learning level. *Neurocomputing* 176: 14–25.

Garcia, L. P. F.; Lorena, A. C.; de Souto, M. C. P.; and Ho, T. K. 2018. Classifier Recommendation Using Data Complexity Measures. In *2018 24th International Conference on Pattern Recognition (ICPR)*, 874–879.

García, S.; Luengo, J.; and Herrera, F. 2016. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems* 98: 1–29.

Guyon, I.; Chaabane, I.; Escalante, H. J.; Escalera, S.; Jajetic, D.; Lloyd, J. R.; Macià, N.; Ray, B.; Romaszko, L.; Sebag, M.; Statnikov, A.; Treguer, S.; and Viegas, E. 2016. A Brief Review of the ChaLearn AutoML Challenge: Anytime Any-dataset Learning Without Human Intervention. In *Workshop on Automatic Machine Learning*, 21–30.

Ho, T. K. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(8): 832–844.

Horn, F.; Pack, R.; and Rieger, M. 2020. The autofeat Python Library for Automated Feature Engineering and Selection. In *Machine Learning and Knowledge Discovery in Databases*, 111–120.

Katz, G.; Shin, E. C. R.; and Song, D. X. 2016. ExploreKit: Automatic Feature Generation and Selection. *2016 IEEE 16th International Conference on Data Mining (ICDM)* .

Kotthoff, L.; Thornton, C.; Hoos, H. H.; Hutter, F.; and Leyton-Brown, K. 2019. Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA. In *Automated Machine Learning: Methods, Systems, Challenges*, 81–95.

Laadan, D.; Vainshtein, R.; Curiel, Y.; Katz, G.; and Rokach, L. 2019. RankML: a Meta Learning-Based Approach for Pre-Ranking Machine Learning Pipelines. *arXiv:1911.00108 [cs, stat]* .

Longadge, R.; and Dongre, S. 2013. Class Imbalance Problem in Data Mining Review. *arXiv:1305.1707 [cs]* .

Maher, M. M. M. Z. A.; and Sakr, S. 2020. SmartML: A Meta Learning-Based Framework for Automated Selection and Hyperparameter Tuning for Machine Learning Algorithms. In *EDBT: 22nd International Conference on Extending Database Technology*.

Mitchell, T. M., Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill, Inc.

Morais, G.; and Prati, R. C. 2013. Complex Network Measures for Data Set Characterization. In *2013 Brazilian Conference on Intelligent Systems*, 12–18.

Munson, M. A. 2012. A study on the importance of and time spent on different modeling steps. *ACM SIGKDD Explorations Newsletter* 13(2): 65–71.

Muñoz, M. A.; Villanova, L; Baatar, D.; and Smith-Miles, K. 2018. Instance spaces for machine learning classification. *Machine Learning* (107): 109–147.

Olson, R. S.; and Moore, J. H. 2019. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. In *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, 151–160.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12(85): 2825–2830.

Pimentel, B.; and de Carvalho, A. 2018. A New Data Characterization for Selecting Clustering Algorithms Using Meta-Learning. *Information Sciences* 477.

Prodromidis, A.; Chan, P.; Stolfo, S.; and others. 2000. Meta-learning in distributed data mining systems: Issues and approaches. *Advances in distributed and parallel knowledge discovery* 3: 81–114.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1): 81–106.

Reif, M.; Shafait, F.; Goldstein, M.; Breuel, T.; and Dengel, A. 2014. Automatic classifier selection for non-experts. *Pattern Analysis and Applications* 17(1): 83–96.

Rice, J. R. 1976. The Algorithm Selection Problem. In *Advances in Computers*, volume 15, 65–118.

Rivolli, A.; Garcia, L. P. F.; Soares, C.; Vanschoren, J.; and de Carvalho, A. C. P. L. F. 2019. Characterizing classification datasets: a study of meta-features for meta-learning. *arXiv:1808.10406 [cs, stat]* .

Schoenfeld, B.; Giraud-Carrier, C.; Poggemann, M.; Christensen, J.; and Seppi, K. 2018. Preprocessor Selection for Machine Learning Pipelines. *arXiv:1810.09942 [cs, stat]* .

Sluban, B.; Gamberger, D.; and Lavra, N. 2010. Advances in Class Noise Detection. In *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, 1105–1106.

Smith-Miles, K. A. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)* .

Swearingen, T.; Drevo, W.; Cyphers, B.; Cuesta-Infante, A.; Ross, A.; and Veeramachaneni, K. 2017. ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data (Big Data)*, 151–162.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. *arXiv:1208.3719 [cs]* .

Vanschoren, J. 2018. Meta-Learning: A Survey. *arXiv:1810.03548 [cs, stat]* .

Vanschoren, J.; Blockeel, H.; Pfahringer, B.; and Holmes, G. 2012. Experiment databases. *Machine Learning* 87(2): 127–158.

Vanschoren, J.; Rijn, J. N. v.; Bischl, B.; and Torgo, L. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15(2): 49–60.

Vilalta, R.; Giraud-Carrier, C.; and Brazdil, P. 2010. Meta-Learning - Concepts and Techniques. In *Data Mining and Knowledge Discovery Handbook*, 717–731.

Wilson, D. L. 1972. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2(3): 408–421.

Wolpert, D.; and Macready, W. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1): 67–82.

Wolpert, D. H. 1992. Stacked generalization. *Neural Networks* 5(2): 241–259.

Zhang, J.; and Mani, I. 2003. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*.

Zhu, X.; and Wu, X. 2004. Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review* 22(3): 177–210.